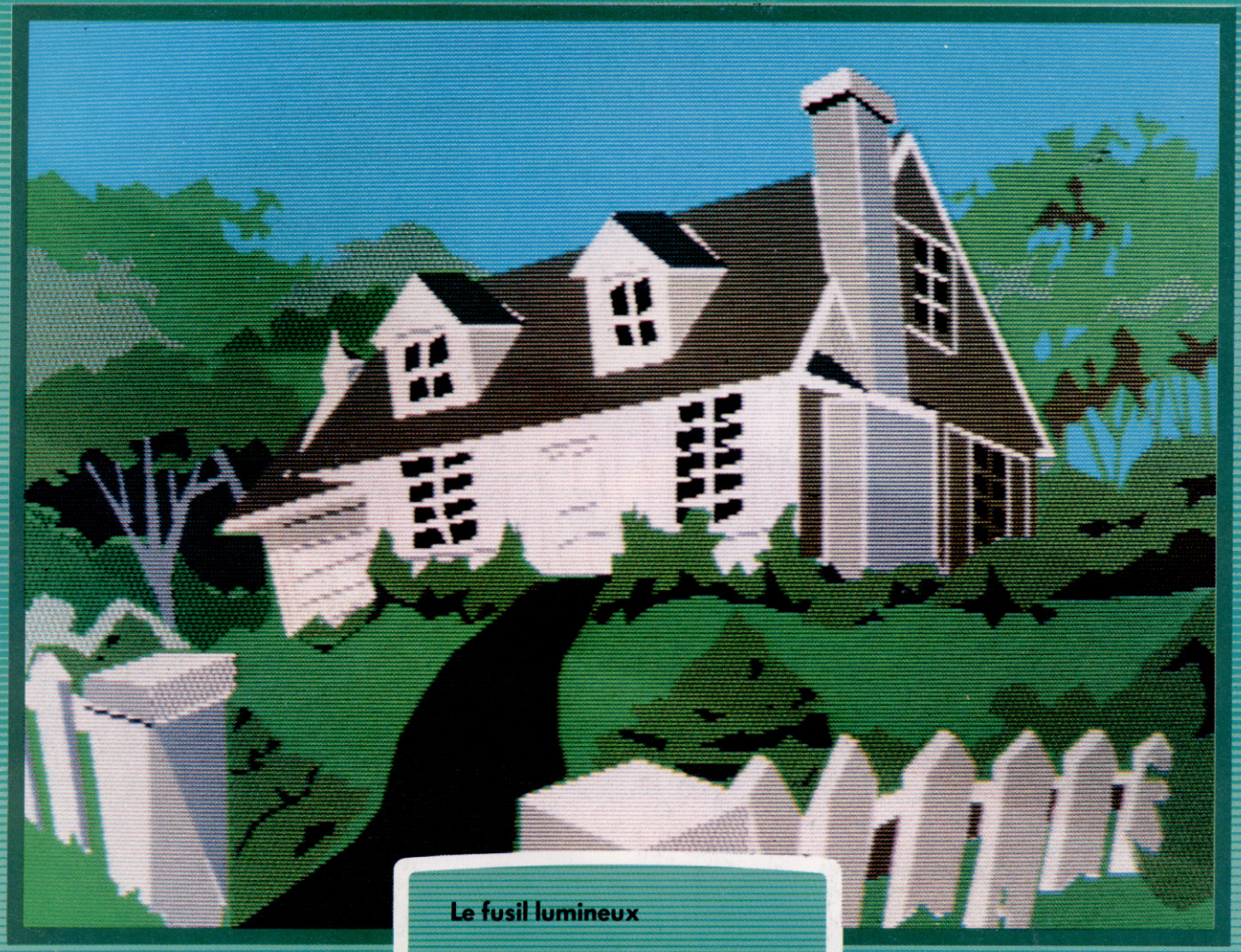
qBC INFORMATIQUE

N° 36

COURS D'INFORMATIQUE PRATIQUE ET FAMILIALE



Le fusil lumineux

Système d'exploitation CP/M

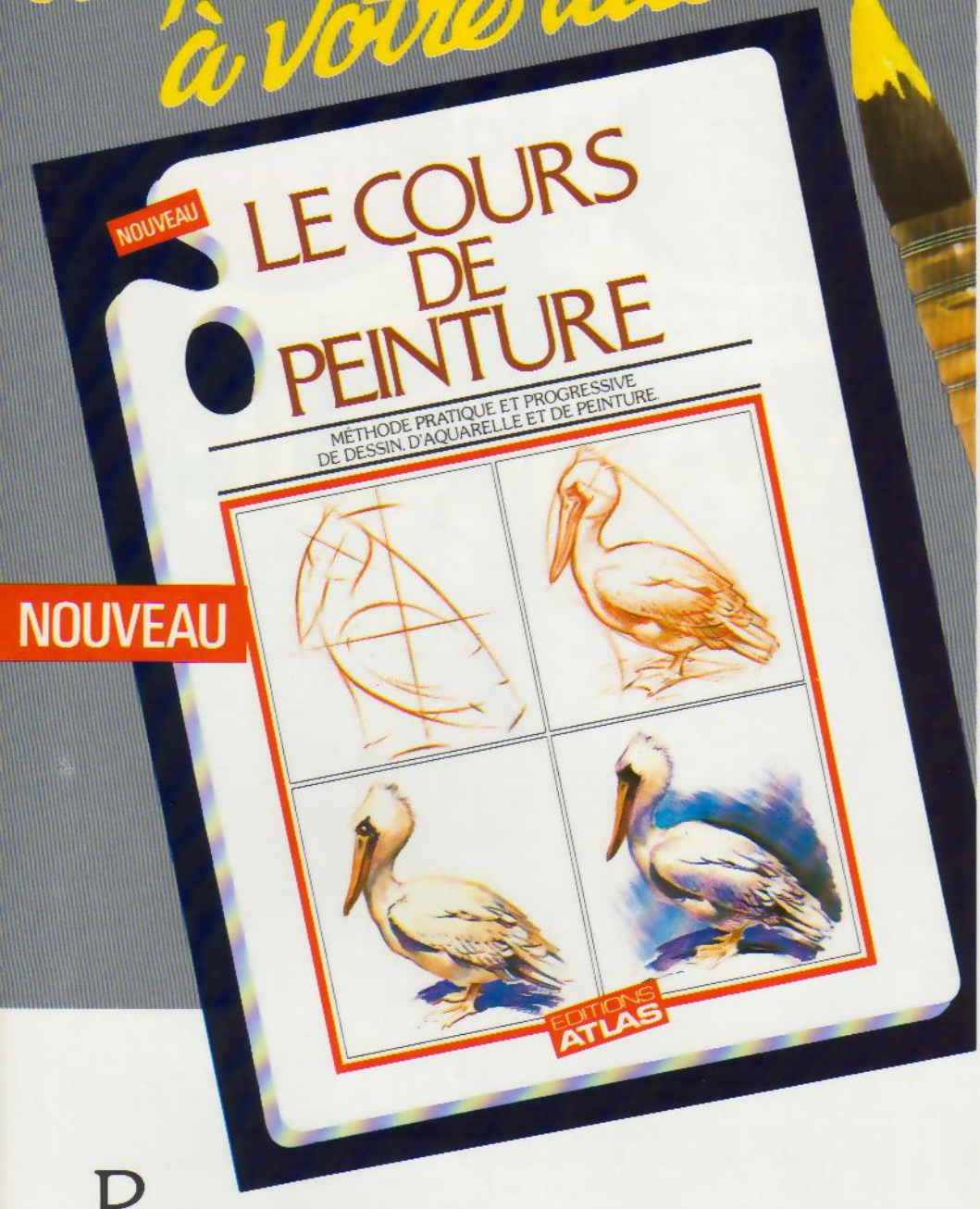
Jeu : le piège

Étude de bascule

EDITIONS ATLAS

M 6062-36-12 F 85 FB - 3,80 FS - $ 1.95

Un fameux coup de pouce à votre talent!



NOUVEAU

Pour tous ceux qui rêvent de savoir peindre un bouquet de roses, un paysage marin, un visage aimé, voici "Le Cours de Peinture" des Editions Atlas.

"Le Cours de Peinture" est un véritable chef-d'œuvre de simplicité : un programme de leçons évolutives imagées qui vous fait passer du trait de crayon du débutant au coup de pinceau de l'artiste.

Avec "Le Cours de Peinture", en quelques semaines, vous apprenez à respecter les formes, à comprendre la perspective, à composer une image, à mélanger les couleurs, à manier le pinceau avec dextérité.

Très vite, vous constatez avec plaisir que vous devenez un véritable artiste, admiré par votre entourage.

UN MAÎTRE À VOS CÔTÉS, POUR EXERCER VOTRE ŒIL, GUIDER VOTRE MAIN, RÉVÉLER VOS DONS.

Robert & Partners

EDITIONS ATLAS

aBc INFORMATIQUE
cours d'informatique pratique et familiale
EDITIONS ATLAS

Édité par ÉDITIONS ATLAS s.a., tour Maine-Montparnasse, 33, avenue du Maine, 75755 Paris *Cedex 15*. Tél. : (37) 35-40-23. Services administratifs et commerciaux : 3, rue de la Taye, 28110 Lucé. Tél. : (37) 35-40-23.

Belgique : ÉDITIONS ATLEN s.a., Bruxelles.

Canada : ÉDITIONS ATLAS CANADA Ltée, Montréal Nord.

Suisse : FINABUCH s.a., ÉDITIONS TRANSALPINES, Mezzovico.

Réalisé par EDENA s.a., 29, boulevard Edgar-Quinet, 75014 Paris.

Direction éditoriale : J.-Fr. Gautier. *Service technique et artistique* : F. Givone et J.-Cl. Bernar. *Iconographie* : J. Pierre. *Correction* : B. Noël.
Publicité : Anne Cayla. Tél. : 202-09-80.

VENTE AU NUMÉRO

Les numéros parus peuvent être obtenus chez les marchands de journaux ou, à défaut, chez les éditeurs, au prix en vigueur au moment de la commande. Ils resteront en principe disponibles pendant six mois après la parution du dernier fascicule de la série. (Pour toute commande par lettre, joindre à votre courrier le règlement, majoré de 10 % de frais de port.)

Pour la France, s'adresser aux services commerciaux des ÉDITIONS ATLAS. Tél. : (37) 35-40-23.

Pour les autres pays, s'adresser aux éditeurs indiqués ci-dessous.

SOUSCRIPTION

Les lecteurs désirant souscrire à l'ensemble de cet ouvrage peuvent s'adresser à :

France : DIFFUSION ATLAS, 3, rue de la Taye, 28110 Lucé. Tél. : (37) 35-40-23.

Belgique : ÉDITIONS ATLEN s.a., 55, avenue Huart-Hamoir, 1030 Bruxelles. Tél. : (02) 242-39-00. Banque Bruxelles-Lambert, compte n° 310-0018465-24 Bruxelles.

Canada : ÉDITIONS ATLAS CANADA Ltée, 11450 boulevard Albert-Hudon, Montréal Nord, H 1G 3J9.

Suisse : FINABUCH s.a., ÉDITIONS TRANSALPINES, zona industriale 6849 Mezzovico-Lugano. Tél. : (091) 95-27-44.

RELIEZ VOS FASCICULES

Des reliures mobiles permettant de relier 12 fascicules sont en vente chez votre marchand de journaux.

ATTENTION : ces reliures, présentées sans numérotation, sont valables indifféremment pour tous les volumes de votre collection. Vous les numéroterez vous-même à l'aide du décalque qui est fourni (avec les instructions nécessaires) dans chaque reliure.

En vente tous les vendredis. Volume III, n° 36.

ABC INFORMATIQUE est réalisé avec la collaboration de Trystan Mordrel *(secrétariat de rédaction)*, Jean-Pierre Bourcier *(coordination)*, Patrick Bazin, Jean-Paul Mourlon, Claire Rémy *(traduction)*, Ghislaine Goullier *(fabrication)*, Marie-Claire Jacquet *(iconographie)*, Patrick Boman *(correction)*.
Crédit photographique, couverture : Maillac-Réa.

Directeur de la publication : Paul Bernabeu. Imprimé en Italie par I.G.D.A., Officine Grafiche, Novara. Distribution en France : N.M.P.P. Tax. Dépôt légal : septembre 1984. 21849. Dépôt légal en Belgique : D/84/2783/27.

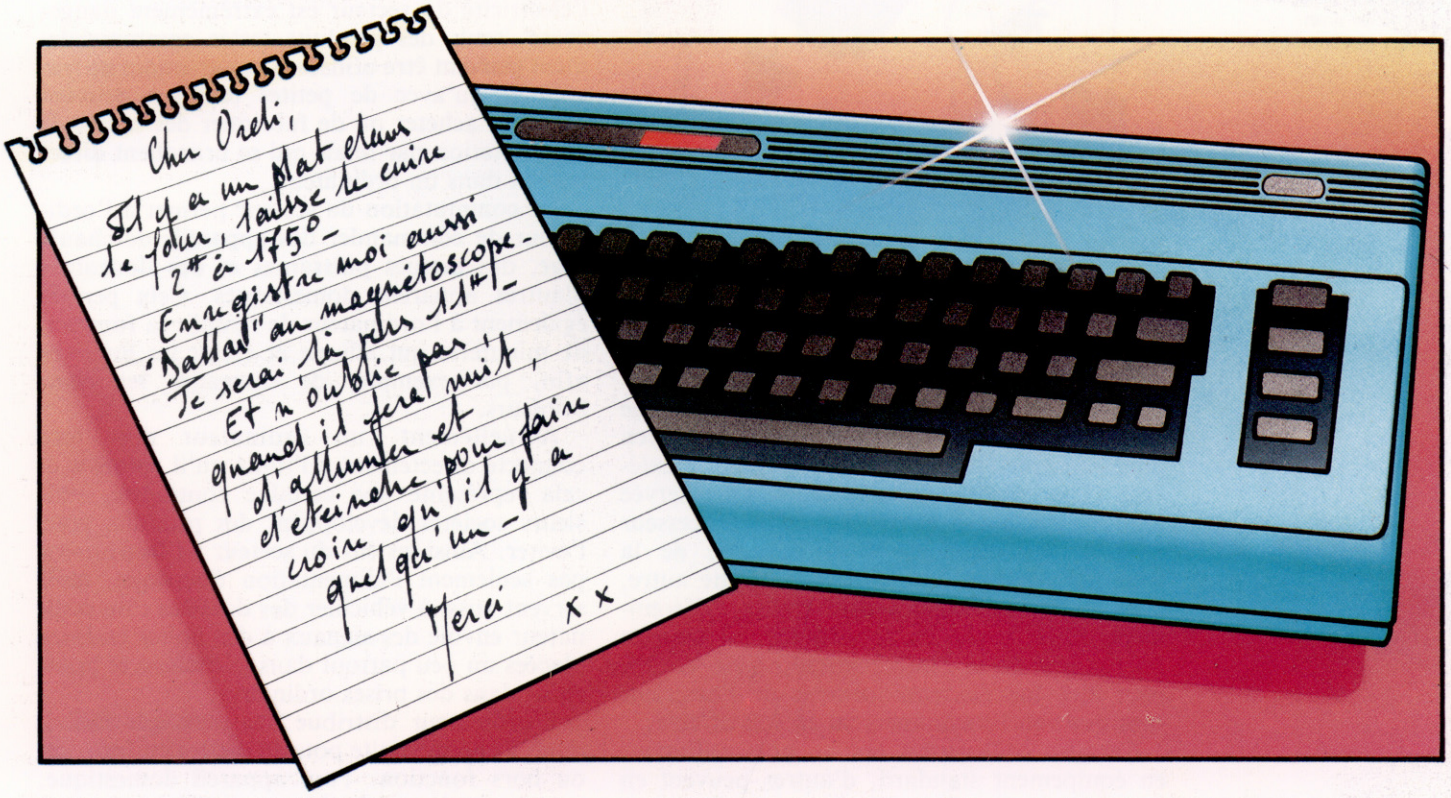
A NOS LECTEURS

En achetant chaque semaine votre fascicule chez le même marchand de journaux, vous serez certain d'être immédiatement servi, en nous facilitant la précision de la distribution. Nous vous en remercions d'avance.

Les Éditions Atlas

Commande de périphérique

Il est possible de trouver sur le marché des composants pour mettre au point un dispositif commandé par ordinateur et qui exécute exactement ce que vous désirez.



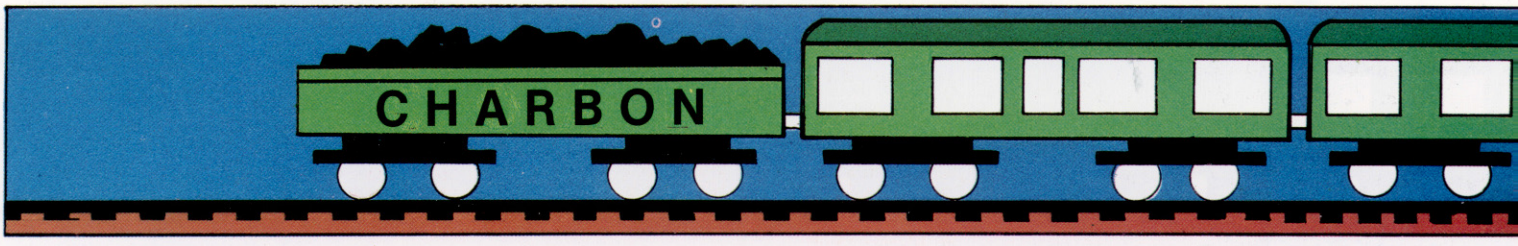
Il n'y a rien de mal à s'amuser avec son ordinateur. Après tout, le faire arroser les plantes, quand vous êtes en vacances, ou ouvrir les rideaux le matin est certainement un grand plaisir.

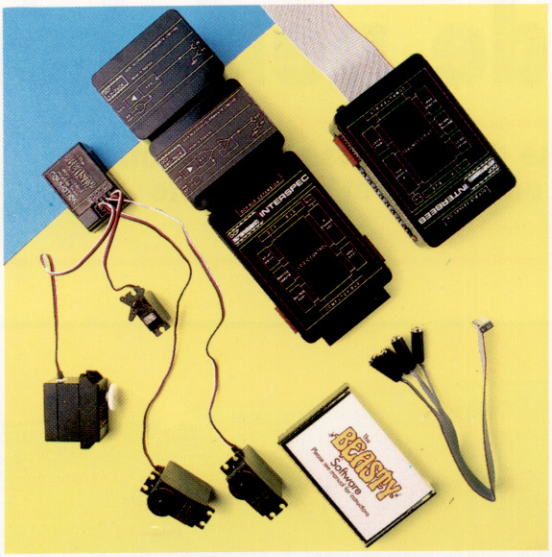
Actuellement, construire ses propres périphériques relève plutôt du jeu-gadget, mais cette activité pourrait à long terme se révéler de plus en plus avantageuse. Plusieurs personnes sont d'avis que les robots ou autres dispositifs commandés par ordinateur deviendront bientôt partie intégrante de notre vie. Et n'oublions pas que des sociétés comme Apple et Atari ont été créées dans les garages d'amateurs qui ne faisaient que s'amuser avec des gadgets et des composants électroniques.

La présence de nombreux éléments est nécessaire dans un système piloté par ordinateur. On a évidemment besoin de l'ordinateur lui-même et du périphérique placé sous son contrôle. L'ordinateur doit aussi être en mesure d'envoyer des messages au dispositif et doit être programmé pour pouvoir décider quels seront les messages. L'ordinateur doit généralement être aussi capable de mesurer l'effet exact de ses interventions afin de pouvoir effectuer des ajustements précis.

Tous les systèmes commandés par ordinateur dépendent des signaux électriques. Malheureusement, les infimes signaux utilisés à l'intérieur d'un ordinateur sont beaucoup trop petits pour être exploités directement. Même un dispositif de la taille d'une ampoule électrique utilise beaucoup plus de puissance que toute pièce interne d'un ordinateur; il est donc nécessaire de trouver un moyen de traduire les petites ten-

Commande à distance
En reliant un ordinateur à divers appareils, il est possible de commander automatiquement leur fonctionnement sous le contrôle d'un programme. L'ordinateur peut intervenir à des moments préprogrammés ou réagir à certaines situations telles qu'une baisse de température ou le déclenchement d'une alarme contre le vol. (Cl. Mike Brownlow.)





Liaisons dangereuses
Des interfaces, directement adaptables à des micro-ordinateurs, sont proposées sur le marché pour assurer un contrôle. Ces unités de type « interrupteur » utilisent des relais. L'ordinateur peut envoyer ou supprimer le courant vers un équipement et, en retour, il peut lire si le capteur est en position *marche* ou *arrêt*.
(Cl. Ian McKinnell.)

sions internes d'un ordinateur en des tensions de plus forte amplitude. Cela est normalement fait en plusieurs étapes.

La première se déroule à l'intérieur même de l'ordinateur. Celui-ci doit trouver un moyen pour envoyer les signaux vers le monde extérieur. Une portion de la mémoire est réservée uniquement à cet usage. Le microprocesseur enverra des messages à cette partie de la mémoire comme il le ferait pour toute autre, mais ici les messages seront traités différemment. Cette partie de la mémoire est appelée *port utilisateur* et l'information qui est stockée peut être lue électroniquement de l'extérieur sans que son fonctionnement en soit affecté.

Certains ordinateurs ont un port utilisateur en équipement standard, d'autres peuvent en être équipés en option. Le port utilisateur peut être utilisé pour commander des diodes électroluminescentes, mais la plupart des systèmes nécessitent également la présence d'autres composants. Le plus pratique est probablement d'ajouter quelques composants électroniques, plus une petite source d'alimentation électrique supplémentaire pour permettre la commande de relais. Les relais sont essentiellement des interrupteurs qui peuvent couper ou faire passer des courants relativement forts et qui peuvent être commandés par des courants faibles. Ils constituent l'une des meilleures façons de convertir les petites impulsions électriques utilisées par un ordinateur en des courants utilisables. L'un des premiers objectifs d'une personne désirant expérimenter la commande par ordinateur doit être d'obtenir un système en mesure d'utiliser des relais. Cela explique le fait qu'il existe une multitude de dispositifs différents commandés par relais : moteurs électriques, pompes à eau, luminaires, cloches, modèles de chemin de fer et modèles automobiles, etc.

La plupart des relais utilisés par des amateurs ne peuvent que piloter des équipements alimentés par piles. Certaines personnes constateront qu'elles ont besoin de plus gros relais pouvant fonctionner sur un courant de secteur. Puisque l'électricité du secteur est extrêmement dangereuse, seuls des produits testés commercialement peuvent être utilisés. Les personnes ne travaillant qu'avec de petites tensions peuvent choisir d'acheter ou de fabriquer des unités de commutation par relais qui se branchent directement dans un ordinateur.

La commutation de secteur permet à l'ordinateur de commander des appareils de chauffage, des lampes puissantes et des douzaines d'autres appareils domestiques. Cela permet également à l'ordinateur de remplir la fonction de minuterie qui allume et éteint les lumières afin, par exemple, de dissuader les cambrioleurs.

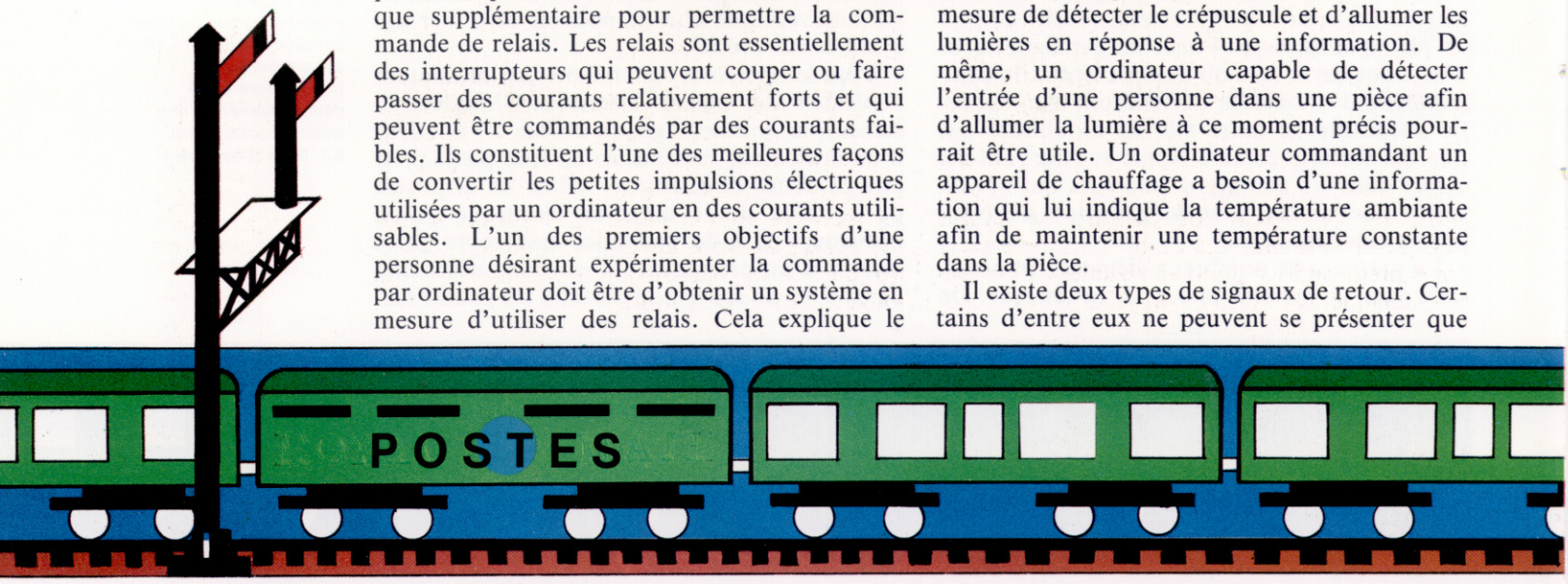
Actuellement, un ordinateur doit être connecté directement à l'unité qu'il contrôle et cela représente une certaine contrainte. Plusieurs sociétés développent des produits pour l'éviter. Ainsi les fils du secteur ne fournissent pas seulement l'alimentation électrique, mais servent aussi à véhiculer des données : un ordinateur envoie des signaux à des unités esclaves placées un peu partout dans la maison et branchées dans des prises ordinaires.

L'ordinateur distribue des messages individuels à chaque unité pour la mettre en fonction ou hors fonction. Tout appareil domestique, comme une lampe de chevet, un téléviseur ou un appareil de chauffage, peut être branché sur l'unité esclave.

Signaux de réaction

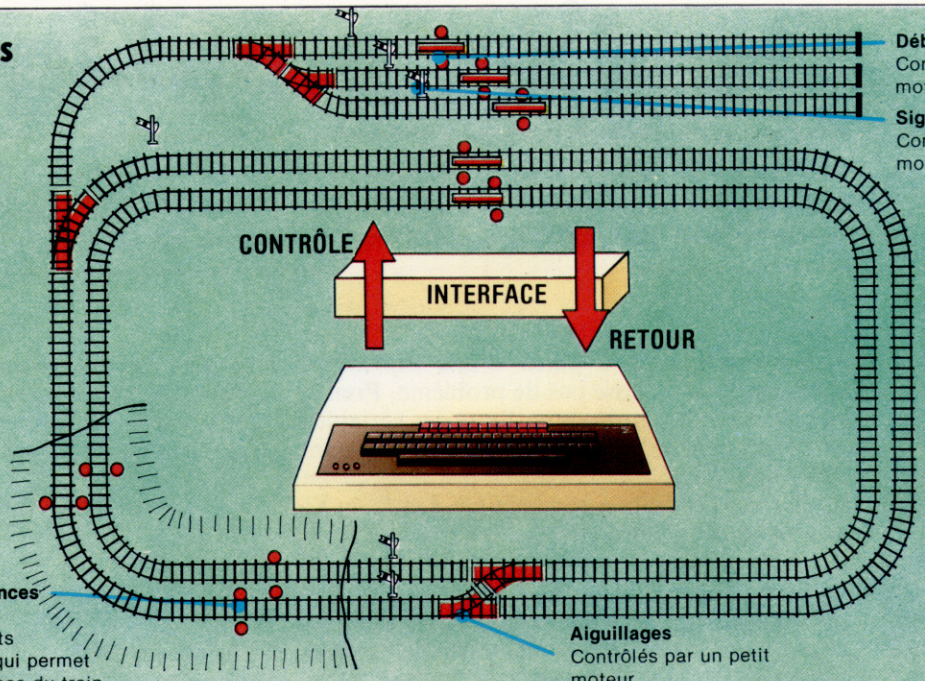
Il serait pratique qu'un ordinateur soit en mesure de détecter le crépuscule et d'allumer les lumières en réponse à une information. De même, un ordinateur capable de détecter l'entrée d'une personne dans une pièce afin d'allumer la lumière à ce moment précis pourrait être utile. Un ordinateur commandant un appareil de chauffage a besoin d'une information qui lui indique la température ambiante afin de maintenir une température constante dans la pièce.

Il existe deux types de signaux de retour. Certains d'entre eux ne peuvent se présenter que



Poste d'aiguillages

Contrôler toutes les pièces d'un dispositif — d'un train électrique comme d'une maison — avec un micro part d'un même principe de base. Une boucle est constituée de façon que le micro envoie un signal, *via* l'interface, à l'objet désigné. Celui-ci retourne une information fournie par ses capteurs (cellules photoélectriques, etc.). Ce retour ferme la boucle et permet à l'objet d'être précisément guidé par l'ordinateur.



Débrayage du train
Contrôlé par un petit moteur.

Signaux
Contrôlés par un petit moteur.

DEL/paire de résistances lumineuses
Ces deux composants forment un capteur qui permet de détecter la présence du train.

Aiguillages
Contrôlés par un petit moteur.

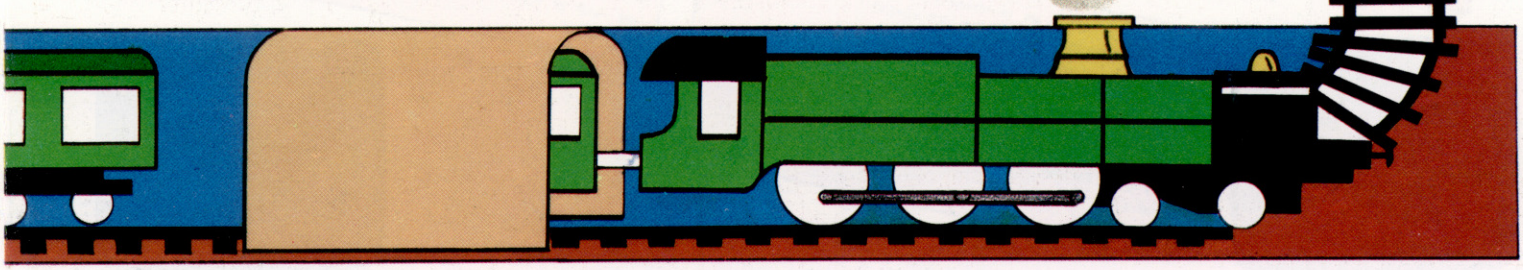
Kevin Jones

sous deux états, sans état intermédiaire. De tels signaux pourraient être produits par un interrupteur qui peut indiquer si une fenêtre est ouverte ou fermée, ou si le bouton d'une sonnette est pressé. Les ports utilisateurs sont capables de lire ces signaux du type marche/arrêt.

Il existe des réactions plus utiles et légèrement plus compliquées qui donnent un signal analogique. Un tel signal peut prendre toute une gamme de valeurs et peut être utilisé pour mesurer une température, une distance relative, un poids ou le niveau de tension d'une pile. Le dispositif qui peut accepter ce type de signal se nomme un convertisseur analogique-numérique, ainsi appelé parce qu'il reçoit de l'équipement contrôlé un ensemble de valeurs variables (le signal analogique) et convertit ces valeurs sous une forme numérique que l'ordinateur peut comprendre.

Un système susceptible de recevoir des signaux de retour est à même de commander une gamme beaucoup plus importante d'appareils. Si un moteur actionnait une roue, l'ordinateur pourrait être en mesure d'estimer la distance parcourue par la roue à un moment précis. Cependant, cela ne fonctionnerait pas si une charge était placée sur la roue ou si les piles entraînant le moteur commençaient à faiblir, puisque la roue tournerait alors plus lentement. Un capteur optique pourrait signaler à l'ordinateur chaque révolution complète de la roue. L'ordinateur pourrait ainsi être en mesure de calculer exactement la distance parcourue, même si la vitesse était variable.

Certains moteurs électriques de conception spéciale disposent d'un dispositif de retour intégré. Cela signifie que l'ordinateur demande de gagner une position particulière et le moteur continue à fonctionner jusqu'à ce qu'il ait atteint cette position. Ces moteurs forment deux catégories : les moteurs pas-à-pas et les servomoteurs. Un moteur pas-à-pas peut tourner continuellement, comme un moteur ordinaire, et il peut être arrêté sur toute position. Cependant, il ne dispose pas d'énormément de puissance et ne peut soulever que de petites charges. Les servomoteurs sont puissants mais ne peuvent effectuer qu'une rotation de quelques degrés — généralement un peu plus de 90 degrés —, d'où un mouvement saccadé. Les moteurs pas-à-pas et les servomoteurs requièrent des unités de commande spéciales disponibles principalement pour des marques d'ordinateurs professionnels. Les servomoteurs sont utilisés dans beaucoup de bras-robots. Il existe de nombreux bras-robots qui peuvent être commandés par un ordinateur domestique, mais ils sont très chers. Il est possible de les construire à un moindre coût à partir de quelques servomoteurs. Mais vous devez, pour réussir, combiner de bonnes connaissances techniques et de longues heures de travail.



Andy Leslie

Création graphique

Nous allons étudier la structure d'un fichier série et sa gestion par le système d'exploitation. Arrêtons-nous également sur les méthodes d'utilisation des fichiers séquentiels dans un programme.

Tracer une courbe à partir d'une expression mathématique ne pose pas de problème. Prenez un ensemble de valeurs pour l'une des variables de l'expression et vous en déduirez les valeurs correspondantes pour l'autre variable. Une courbe mathématique aussi simple que $Y = X^2$ peut se traduire ainsi par une table :

X	-5	-4	-3	-2	-1	0	1	2	3	4	5
Y	25	16	9	4	1	0	1	4	9	16	25

En traçant sur du papier millimétré les points ainsi obtenus et en les reliant, nous obtenons la courbe de l'expression mathématique. La trajectoire est également appelée *lieu géométrique*. En manipulant et en combinant plusieurs lieux géométriques, on obtient des effets graphiques étonnants et sans grand effort d'imagination.

Étudions à cette fin un lieu géométrique bien connu, le cercle.

Les éléments X et Y de l'équation sont en effet décrits selon une troisième variable, ou paramètre. En faisant varier cette dernière selon un ensemble de valeurs, il est possible d'établir des coordonnées (X,Y) :

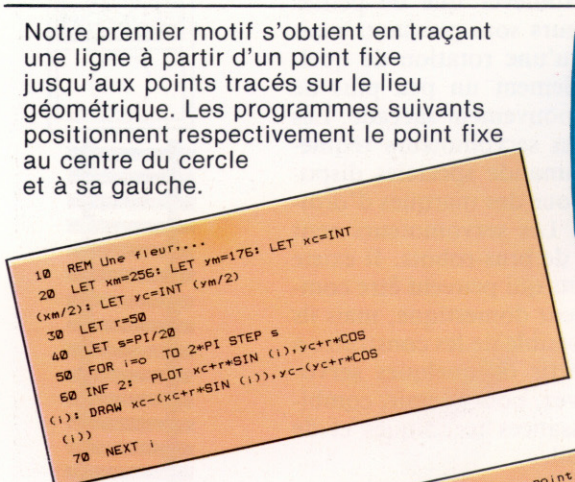
X = R SIN (I)
Y = R COS (I)

Si nous élaborons des valeurs pour X et Y selon l'accroissement de l'angle I (de 0 à 360°), nous obtenons le lieu géométrique d'un cercle. Le court programme suivant fait pour le Spectrum devra créer un cercle (voir « Variantes de BASIC » pour les autres versions).

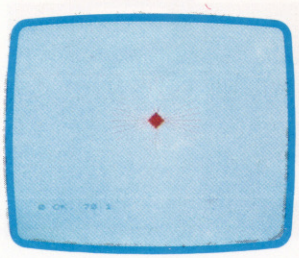
```
10  REM Tracé d'un cercle
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  INK 2: PLOT xc+r*SIN (i),yc+r*COS
(i)
70  NEXT i
```

Vous remarquez qu'en modifiant la valeur de STEP (distance entre deux points), vous faites varier la définition du cercle, et vous changez la vitesse à laquelle il est tracé. La plupart des versions du BASIC ne sont pas assez rapides pour tracer des cercles réguliers à partir de nombreux points à une vitesse acceptable. Pour y remédier, il est souvent préférable d'utiliser un certain nombre de lignes droites qui relient les points. Si ces dernières sont suffisamment nombreuses, vous parviendrez à un compromis valable entre vitesse d'exécution du cercle et régularité du tracé. La même formule peut également servir à tracer des arcs (prenez diverses valeurs pour I) et des ellipses (donnez à R, une valeur différente dans X et dans Y).

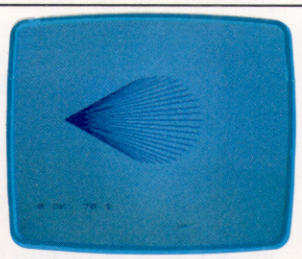
Notre premier motif s'obtient en traçant une ligne à partir d'un point fixe jusqu'aux points tracés sur le lieu géométrique. Les programmes suivants positionnent respectivement le point fixe au centre du cercle et à sa gauche.



```
10  REM Une fleur...
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  INK 2:  PLOT xc+r*SIN (i),yc+r*COS
(i): DRAW xc-(xc+r*SIN (i)),yc-(yc+r*COS
(i))
70  NEXT i
```

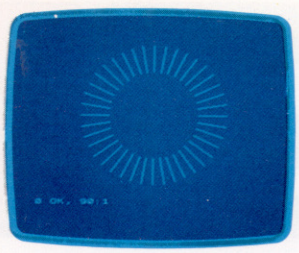


```
10  REM Le Cercle et le point fixe
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  INK 2:  PLOT xc+r*SIN (i),yc+r*COS
(i): DRAW xc-100-(xc+r*SIN (i)),yc-(yc+r
*COS (i))
70  NEXT I
```

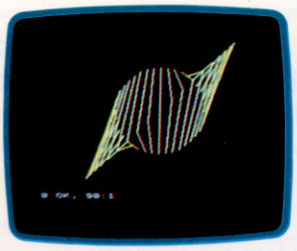


Nous utilisons ensuite un point changeant et non plus un point fixe. Nous tracerons simultanément deux lieux géométriques ainsi que des lignes reliant les points correspondants des deux cercles.

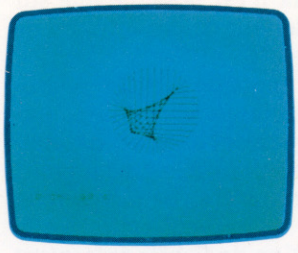
```
10  REM Cercles imbriqués
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  LET x=xc+(r+30)*SIN (i): LET y=yc+
(r+30)*COS (i)
70  LET p=xc+r*SIN (i): LET q=yc+r*COS
(i)
80  INK 2: PLOT x,y: DRAW p-x,q-y
90  NEXT i
```



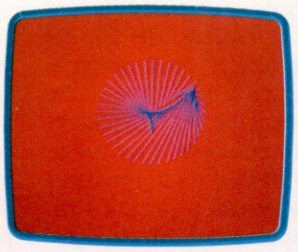
Le programme que nous avons développé dispose maintenant de suffisamment d'informations pour tracer des centaines de motifs. Pour cela, il nous faut faire varier l'un des lieux géométriques ou les deux. Une variante simple peut consister simplement à permuter SIN et COS.



```
10  REM Cercles tournant
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  LET x=xc+(r+30)*COS (i): LET y=yc+
(r+30)*SIN (i)
70  LET p=xc+r*SIN (i): LET q=yc+r*COS
(i)
80  INK 2: PLOT x,y: DRAW p-x,q-y
90  NEXT i
```



Les variations sur cette base sont multiples. Un manuel classique de mathématiques vous fournira d'autres formules de courbes. Mais vous trouverez sans doute qu'il est plus enrichissant d'expérimenter par vous-même de nouveaux programmes. Ainsi, quelques modifications très simples au programme de base vous feront découvrir d'autres possibilités. A ce niveau, c'est l'ordinateur qui vous donnera des idées !



```
10  REM Cercle & SIN
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  LET x=xc+SIN (i)*80: LET y=yc+(r+3
0)*SIN (i)
70  LET p=xc+r*SIN (i): LET q=yc+r*COS
(i)
80  INK 2: PLOT x,y: DRAW p-x,q-y
90  NEXT i
```

```
10  REM Cercle & SIN*COS
20  LET xm=256: LET ym=176: LET xc=INT
(xm/2): LET yc=INT (ym/2)
30  LET r=50
40  LET s=PI/20
50  FOR i=0  TO 2*PI STEP s
60  LET x=xc+r*SIN (i)*COS (i): LET y=
yc+r*SIN (i)*COS (i)
70  LET p=xc+r*SIN (i): LET q=yc+r*COS
(i)
80  INK 2: PLOT x,y: DRAW p-x,q-y
90  NEXT i
```

```
10  REM Motifs de cercles aléatoires
15  RANDOMIZE
20  LET xm=256: LET ym=176: LET xc=INT
(x=/2): LET yc=INT (ym/2)
30 LET r=60: LET s=PI/20
35 CLS
40  LET c=INT (RND*4)+1: LET d=INT (RN
(D*4)+1: LET e=INT (RND*4)+1 LET f=INT (
RND*4)+1
50  FOR i=0  TO 2*PI STEP s
60  LET x=xc+r*SIN (i/c)*COS (i*d): LE
T y=yc+r*SIN (i/e)*COS (i*f)
70  LET p=xc+r*SIN (i): LET q=yc+r*COS
(i)
80 PLOT x,y: DRAW p-x,q-y
90  NEXT i
100 IF INKEY$="" THEN GO TO 100
110 GO TO 35
```

Variantes de basic

Les programmes listés ici sont destinés à un Spectrum 16 K ou 48 K. Mais leur transcription pour d'autres machines est très simple à obtenir. Votre micro doit comporter le mode graphique haute résolution (de préférence, au moins 256 × 176), un BASIC avec virgule flottante doté des fonctions SIN et COS, ainsi qu'une commande pour tracer des points (PLOT), et des lignes (DRAW).
Les premières adaptations nécessaires concernent XM et YM, les valeurs maximales de X et Y pouvant être attribuées à votre machine. Selon les fonctions utilisées, vous découvrirez que d'autres constantes du programme, comme R ou S par exemple, devront être changées. Vous devez ensuite vérifier que votre micro dispose du mode graphique approprié, et choisir une couleur de tracé. Il vous faut en dernier lieu une commande pour tracer des lignes entre les coordonnées (X, Y), et P et Q.

Le BBC MICRO
Tous les modes du BBC utilisent une grille de 1280 × 1024 pour les tracés et vous découvrirez que MODE 0 produit des résultats spectaculaires. GCOL déterminera la couleur de tracé, et DRAW tracera les lignes.

DRAGON 32/64
Le PMODE 4 du Dragon présente une grille de 256 × 192. Utilisez respectivement SCREEN 1, 0 ou SCREEN 1, 1 pour obtenir une couleur de fond verte ou ambre.
La commande LINE peut servir à tracer les lignes (LINE(X,Y)–(P,Q),PSET).

COMMODORE 64/VIC-20
Ces machines disposent de modes graphiques haute résolution pouvant tout à fait convenir, mais elles ne comportent malheureusement pas les commandes appropriées. Pour exécuter les programmes, vous devrez donc soit créer vos propres commandes point et ligne, soit utiliser une cartouche d'extension BASIC telle que la cartouche Simon.

CAMPUTERS LYNX
Le Lynx est tout à fait approprié à ce genre de tâches, du fait qu'il dispose d'un mode d'affichage graphique complet sur 8 couleurs, de définition 256 × 248. Il n'a pas besoin d'une commande de mode pour l'activer. Utilisez INK pour choisir la couleur du tracé, et MOVE et DRAW pour les lignes.

ORIC 1/ATMOS
HIRES active l'affichage graphique sur 240 × 200 de l'Oric. Les lignes peuvent être obtenues avec CURSET pour se déplacer sur leur point d'origine (X,Y), et ensuite avec DRAW pour le tracé. Sur Oric, DRAW est relative, aussi cette commande doit être sous la forme DRAW p-x,q-y.

Idées de motifs

1. En revenant à l'idée d'une simple boucle pour tracer un cercle, nous avons décrit comment obtenir des arcs et des ellipses à partir du même programme. Voyez maintenant si vous pouvez tracer des spirales.
2. Essayez d'autres fonctions pour tracer des lieux géométriques, par exemple SQR ou TAN. Méfiez-vous pourtant de ces fonctions qui ont tendance à générer des nombres difficilement maniables. Vous pouvez néanmoins espérer des résultats intéressants.
3. Créez des versions animées de ces programmes. En utilisant des tableaux pour enregistrer les cinq dernières lignes tracées, vous pouvez montrer un ensemble de cinq lignes se poursuivant autour de deux cercles.
4. Et pourquoi ne pas créer des motifs fondés sur la présence simultanée de trois lieux géométriques ? Faites en sorte que parmi ceux-ci, deux soient très simples (un cercle et une ligne, peut-être), afin de garder une image lisible.

Fichiers en séquentiel

Nous abordons l'étude de quelques routines intéressantes que l'on peut créer en quelques lignes de code basic. Ce premier article montre comment faire de véritables motifs élaborés.

Un fichier séquentiel est constitué d'un bloc indivisible de données résidant sur disquette ou sur cassette. L'accès et la mise à jour d'un tel fichier posent problème. Ainsi, pour accéder à un article du fichier, vous devez passer par tous les enregistrements précédents. Pour faire une mise à jour, il faut généralement faire une copie jusqu'au point où les modifications sont nécessaires, ces dernières étant alors liées au nouveau fichier. Le fichier originel finit ensuite d'être copié.

Il est important de noter que l'information doit être organisée de manière rigoureuse à l'intérieur du fichier. Le choix d'organisation revient au programmeur et dépend de l'application. Si le fichier doit contenir un texte, il sera simplement constitué d'une suite de codes ASCII terminée par un marqueur de fin de fichier. En revanche, si le fichier doit contenir une base de données telle qu'un catalogue de livres, l'information devra être organisée de manière appropriée. La manière habituelle consiste à diviser le fichier en enregistrements et en zones. Pour l'exemple d'un répertoire de livres, chaque livre correspondra à un enregistrement (une entrée au fichier). Chaque enregistrement comportera un certain nombre de zones, telles que le titre, l'auteur, l'éditeur, etc. A l'intérieur du fichier séquentiel, ces distinctions seront matérialisées par des caractères spéciaux de séparation entre les divers postes de données.

Le marqueur de zones et d'enregistrements est habituellement un caractère « retour chariot » (numéro de code ASCII, 13). Comme le fichier comporte toujours le même nombre de zones par enregistrement, il est facile pour le programme de savoir où se termine un enregistrement et où commence le suivant.

Une fois un fichier série créé, vous devez pouvoir y accéder et le mettre à jour. Les opérations de base sur les fichiers sont : restitution d'un enregistrement (lecture), concaténation d'enregistrements, effacement et modification d'enregistrements (édition). Les organigrammes suivants indiquent les diverses procédures pour ces opérations. Comme vous ne pouvez lire un fichier séquentiel que dans l'ordre, et que vous n'êtes pas libre d'y apporter directement des modifications, ces opérations passent par des copies : lecture et copie simultanées. Toute information nouvelle est ainsi apportée lorsqu'elle se présente, et une nouvelle version du fichier est alors créée. La version précédente est soit abandonnée, soit conservée en tant que copie de sauvegarde. Ces techniques simples sont les principes de base de toutes routines d'organisation de fichier séquentiel. Elles supposent une certaine configuration du système d'exploitation. Il doit y avoir en effet deux fichiers ouverts en même temps, l'un en lecture, l'autre en écriture. Cela n'est possible que sur des systèmes à double lecteur de cassettes ou de disquettes.

C'est la raison pour laquelle certains micro-ordinateurs ont des interfaces pour double lecteur de cassettes. Les systèmes avec lecteur simple se limitent à des fichiers suffisamment petits pour être entièrement transférés en mémoire et y être gérés.

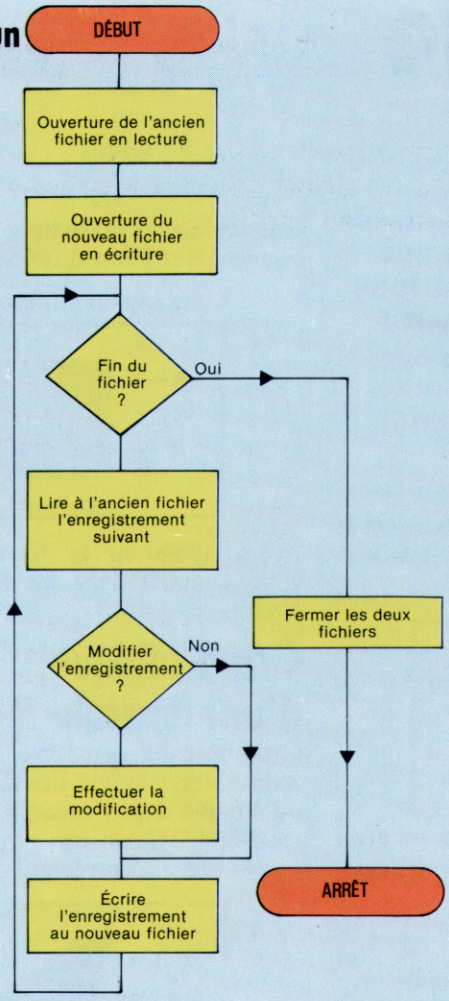
Ces méthodes de gestion de fichier présentent en outre l'avantage de générer à chaque mise à jour (additions, suppressions ou modifications de fichiers) une nouvelle version, tout en conservant le fichier originel. Il est de pratique courante de garder les deux fichiers, de sorte que si quelque chose arrivait au fichier actualisé, on pourrait toujours récupérer la version précédente. On dit alors que cette dernière a une génération de retard. La plupart des informaticiens gardent toujours les trois dernières générations de mises à jour : la version courante ou version de troisième génération, la version précédente dite de deuxième génération, et celle d'avant, de première génération.

Ces techniques conviennent à des fichiers trop grands pour tenir dans la mémoire de l'ordinateur. En effet, une partie seulement du fichier (quelques enregistrements) est chargée dans le même temps en mémoire pour y être traitée. Cependant, avec de petits fichiers, il est possible d'optimiser les performances de traitement, en mettant la totalité du fichier, sous forme de tableaux, en mémoire. Toutes les opérations peuvent alors avoir lieu à très haute vitesse avant de récrire le nouveau fichier sur disque ou sur cassette.

Cette dernière approche présente cependant un danger important : les modifications ne deviennent permanentes que lorsque l'information est récrite. Cela signifie que les données peuvent être perdues en cas de défaillance de l'ordinateur ou du programme, ou si l'on arrête l'ordinateur au cours du fonctionnement. Si vous utilisez de tels programmes, ayez soin de faire fréquemment des copies du fichier à sauvegarder. Assurez-vous qu'une copie courante a été faite avant que le programme s'achève.

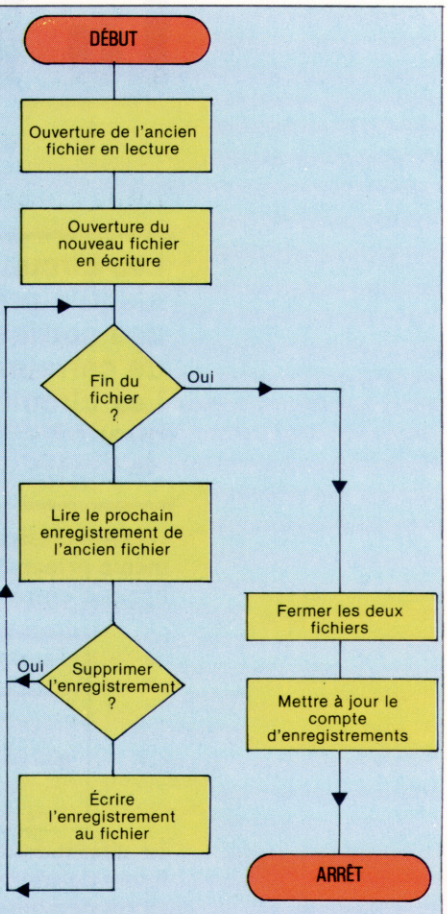
Modification d'un enregistrement

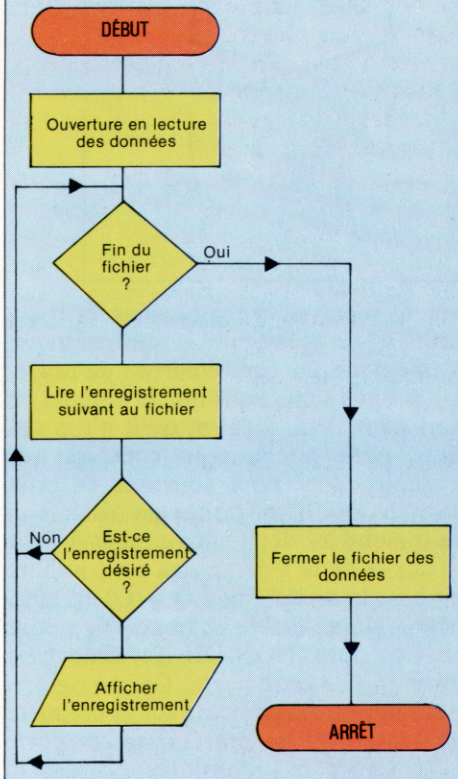
La modification d'un enregistrement s'obtient de la manière suivante : les enregistrements précédant la modification sont recopiés sur un nouveau fichier. L'enregistrement à modifier est transmis en mémoire vive où il est mis à jour. Il est ensuite recopié sur le nouveau fichier. Les enregistrements suivants de l'ancien fichier sont alors recopiés à sa suite. Une lecture de fichier autorise autant de modifications qu'on le désire.



Suppression d'un enregistrement

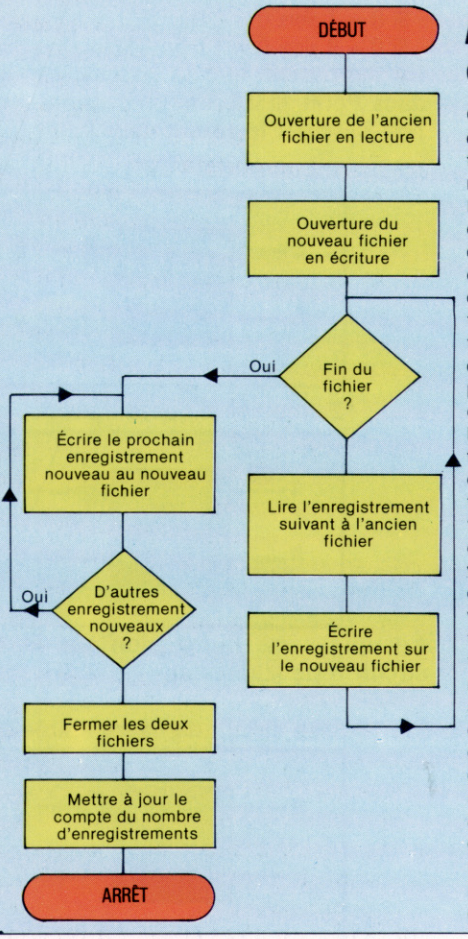
Pour effectuer une suppression, il faut recopier le fichier jusqu'à l'enregistrement concerné, sans qu'il soit recopié en mémoire. Le restant du fichier d'origine est alors recopié à la suite sur le nouveau fichier. Plusieurs enregistrements peuvent être supprimés en une seule fois. Tout comme pour la modification, le compte du nombre d'enregistrements doit être scrupuleusement tenu après intervention.





Transmettre un enregistrement

Les fichiers séquentiels ne conviennent pas aux applications où il faut pouvoir extraire les anciens enregistrements. Chaque fois que vous voulez chercher un enregistrement, il vous faut lire séquentiellement la totalité du fichier jusqu'à ce que vous le rencontriez. Cela représente une perte de temps considérable. Si vous recherchez un enregistrement parmi une liste de noms, votre programme fera une boucle, examinant chaque enregistrement jusqu'à ce qu'il trouve le bon. Si vous cherchez plusieurs enregistrements, ils peuvent être lus à la suite, pourvu qu'ils figurent dans l'ordre où ils se trouvent au fichier.



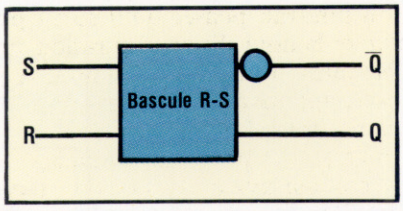
Ajouter un enregistrement

On ajoute un enregistrement à un fichier de deux manières. Certains BASIC ont une commande APPEND (lier) qui vous permet de concaténer un enregistrement à un fichier (l'ajouter à la fin du fichier). Dans le cas de fichiers ne pouvant utiliser cette commande, il faut lire le fichier dans sa totalité et en faire une copie. Au lieu de fermer le nouveau fichier une fois entièrement recopié, vous lui ajoutez les enregistrements voulus, avant de fermer les deux fichiers. Il est nécessaire de tenir à jour le compte du nombre d'enregistrements. Si cette routine de mise à jour est stockée avec le fichier, elle doit s'assurer que la nouvelle valeur est immédiatement écrite au fichier.

La bascule

Les circuits que nous avons vus jusqu'à présent produisaient tous des sorties déterminées à partir de certains signaux en entrée. Les circuits séquentiels, quant à eux, génèrent un signal de sortie stable.

Il existe plusieurs sortes de bascules basées sur le même principe. La bascule R-S comporte deux lignes d'entrée et deux lignes de sortie.

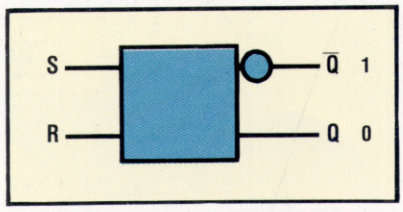


Le circuit est conçu de telle sorte que les deux lignes de sortie, Q et $\overline{Q}$, soient toujours en opposition de phase. C'est-à-dire :

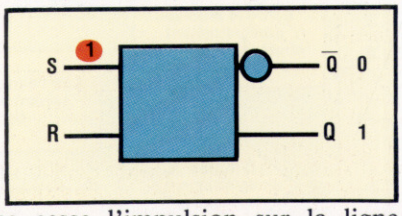
Si Q = 1 alors $\overline{Q}$ = 0 (état SET)
Si Q = 0 alors $\overline{Q}$ = 1 (état RESET)

En supposant que la bascule est initialement dans l'état RESET, une impulsion sur la ligne S fera basculer le circuit dans l'état SET.

1. État initial (RESET)

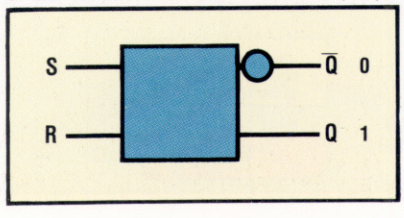


2. Impulsion sur la ligne SET



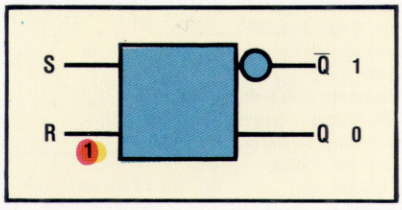
Lorsque cesse l'impulsion sur la ligne S, le circuit reste stable dans l'état SET.

3. Le circuit demeure stable (SET)



Une impulsion sur la ligne R bascule le circuit de nouveau sur l'état d'origine, RESET.

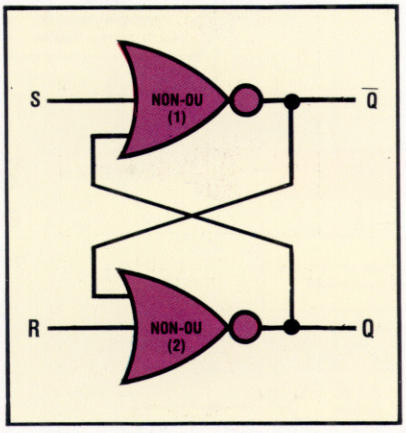
4. Impulsion sur la ligne RESET



Après avoir vu la fonction d'une bascule, voyons maintenant les éléments logiques du circuit.

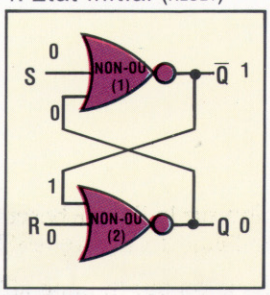
Circuit logique d'une bascule R-S

Cette bascule peut être obtenue par diverses techniques, comme lier deux portes NON-ET, ou comme dans l'exemple donné ici, deux portes NON-OU, de sorte que la sortie d'une des portes serve d'entrée pour l'autre.

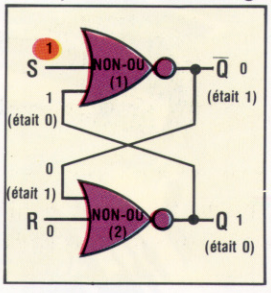


Nous pouvons représenter maintenant les fonctions SET et RESET de la bascule, et voir comment elles sont obtenues par combinaisons de portes NON-OU. Si nous supposons que la bascule est initialement dans l'état RESET et qu'il n'y a pas d'impulsion, nous pouvons en conclure que l'état est stable. Vous vous souvenez en effet qu'une porte NON-OU ne donne un résultat en sortie que lorsque les deux entrées sont 0. Une impulsion sur la ligne S dérangera cet équilibre et transformera la sortie « non Q » ($\overline{Q}$) en zéro. Cela affectera alors l'entrée en boucle de retour de la deuxième porte NON-OU (2), changeant en 1 la sortie (Q) de cette porte. Ce qui signifie alors que si l'impulsion demeure sur la première porte NON-OU (1), les entrées sur la porte NON-OU (1) seront de 1. Ainsi, les sorties de la porte NON-OU (1) resteront à 0.

1. État initial (RESET)

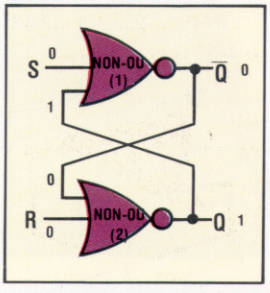


2. Impulsion sur la ligne SET

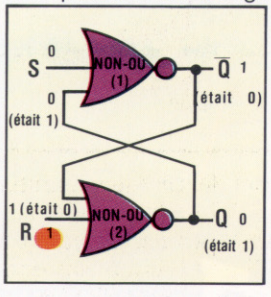


Le circuit reste stable même lorsque cesse l'impulsion sur la ligne S. Lorsqu'on envoie une impulsion sur la ligne R, le circuit redevient instable. Après une démarche similaire à celle précédemment décrite, il passe à nouveau dans l'état RESET et redevient stable.

3. Le circuit reste stable (SET)

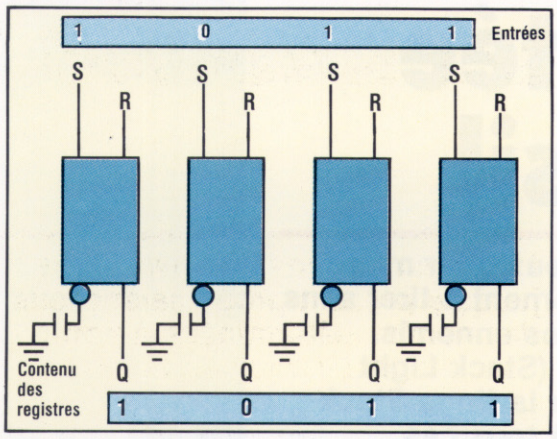


4. Impulsion sur la ligne RESET



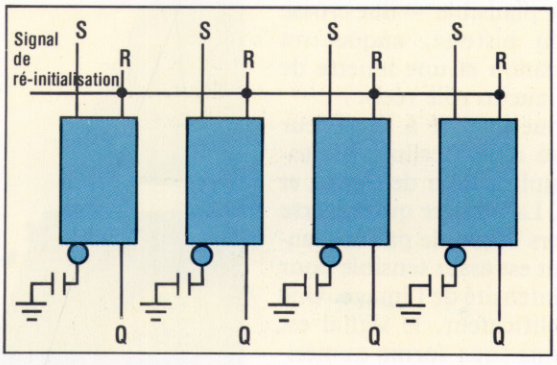
Registres

Votre microprocesseur est en grande partie constitué de registres tels que l'accumulateur, le registre d'instructions et le registre d'index. La plupart des registres peuvent contenir des mots sur 8 bits, c'est-à-dire des nombres binaires compris entre 0 et 255. Du fait que les registres acceptent et retiennent une information binaire, il n'est pas surprenant qu'ils soient constitués de 8 bascules. Pour simplifier, nous allons aborder maintenant le fonctionnement d'un registre à 4 bits.



Vous remarquez dans ces combinaisons que la sortie « non Q » n'est pas utilisée. Les lignes Q produisent un résultat en sortie qui correspond à la configuration de bits utilisée pour les lignes S des bascules. Si nous voulions remplacer le premier nombre du registre, par exemple par 0110, on pourrait penser qu'il suffit pour cela d'appliquer la nouvelle configuration aux lignes S des bascules. Mais en fait, si nous le faisons, nous nous apercevons que le nombre résultant au registre sera 1111.

La solution à ce problème est de mettre à zéro chaque bascule avant de stocker le deuxième nombre. Puisque toutes les bascules doivent être remises à zéro en même temps, il est plus commode de les relier.



Liz Dixon

Par la suite, nous étudierons d'autres circuits séquentiels, parmi lesquels la bascule de type D, et la bascule J-K.

Exercice 7

1. Pourquoi une bascule est-elle également appelée *circuit bistable ?*

2. Lorsqu'on allume un ordinateur, une certaine bascule se trouve dans l'état suivant :

$Q = 0, \overline{Q} = 0, S = 0, R = 0$

a) Cet état est-il stable ?
b) Sinon, pour quel état va-t-il changer ?
c) La bascule peut-elle changer pour un autre état que celui donné par la réponse précédente ? (Suggestion : essayez de commencer par l'autre porte.)
d) Quelle démarche entreprendre, lors de l'allumage de votre ordinateur, pour être sûr de l'état des registres ?

A portée de fusil

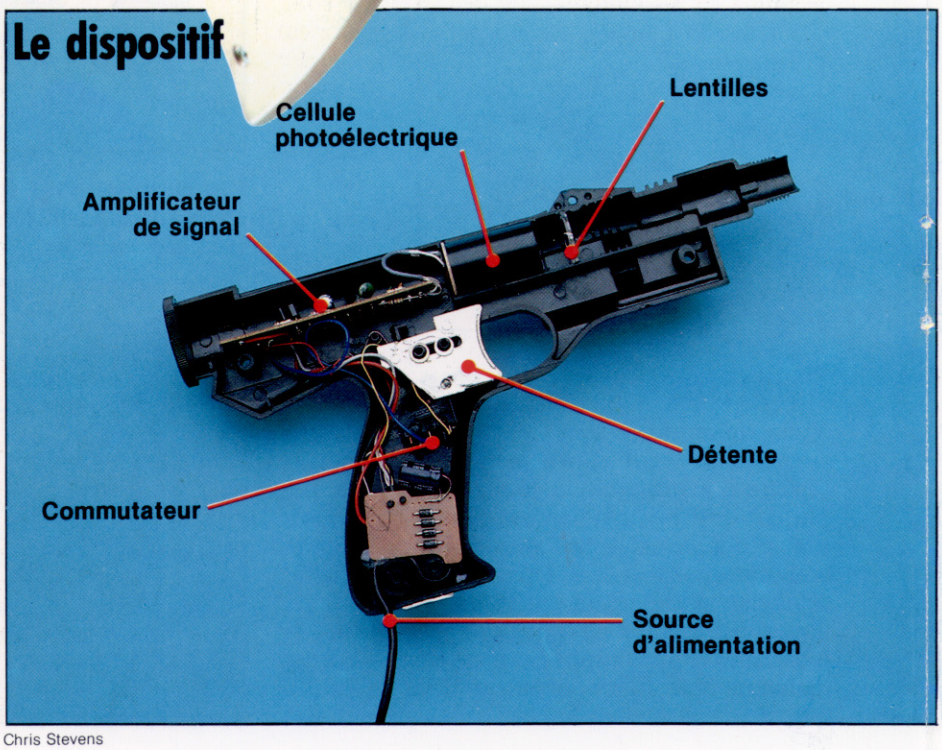
D'innombrables jeux pour micro-ordinateurs se bornent à tirer sans fin sur de multiples ennemis. Le fusil lumineux (Stack Light Rifle) proposé par la firme Stack entend apporter à tous ces programmes un supplément de réalisme. Ce dispositif se comporte comme un crayon optique et permet de se passer d'un clavier ou d'une manette.

L'élément principal de l'ensemble est un pistolet à visée électronique, raccordé à l'ordinateur par une bonne longueur de câble. Selon les cas, le raccord proprement dit s'effectue grâce à une prise ou un connecteur plat. Sur le Spectrum Sinclair, le connecteur comporte deux puces et quelques composants très simples, qui permettent d'interfacer l'appareil et l'arme. Pour que celle-ci soit plus précise — et pour qu'on ait affaire à un fusil d'allure plausible — une crosse est fixée à l'arrière du pistolet, auquel on adjoint de surcroît un canon et une lunette de visée (qui n'a d'ailleurs aucun rôle réel).

Le système électronique installé à l'intérieur du pistolet est composé d'une cellule photoélectrique, d'un petit amplificateur de signal, et d'une mémoire tampon. La lumière qui traverse le canon est polarisée vers la cellule par des lentilles en plastique : le tout est assez sensible pour déceler les variations d'intensité de l'image. Une fois accueilli par l'amplificateur, le signal est écrêté afin d'être transmis sous forme numérique, et non analogique. Il est enfin véhiculé vers l'ordinateur par l'intermédiaire d'un commutateur. Le point de l'écran qui est balayé à ce moment-là est celui que vise le fusil. L'ordinateur reçoit le signal et en compare la valeur avec celle de la cible; si elles concordent, le joueur a réussi à atteindre l'objectif.

Le fusil lumineux Stack existe en plusieurs versions, destinées respectivement au Spectrum, au Vic-20 et au Commodore 64. Toutes fonctionnent selon les mêmes principes. La firme propose aussi trois jeux sur cassette, mais c'est à peu près tout ce qui est disponible. Les firmes productrices de logiciels, qui lancent des jeux tout à fait adaptés à l'emploi de l'engin, ne semblent guère s'y être intéressées, à de rares exceptions près. Cette carence empêche donc le fusil de remplacer le manche à balai, d'autant plus que les détails techniques relatifs à son fonctionnement ne sont pas disponibles.

Le dispositif



Chris Stevens

Pourtant les principes de base sont simples, et l'appareil est une sorte de gros crayon optique, qui serait capable de travailler à distance de l'écran (3 m environ), et non en contact avec lui. Les lentilles et le canon permettent de filtrer la lumière ambiante. Cela rend possible une certaine précision — assez limitée cependant —, et l'utilisateur a tout loisir de faire feu depuis son fauteuil. Les jeux disponibles ne donnent qu'une idée très médiocre des possibilités du fusil; ils n'ont rien de remarquable et leur graphisme reste sommaire.

L'un des gros problèmes, lorsqu'on veut utiliser un crayon optique (et donc sa version géante, le modèle Stack), est de rédiger le programme correspondant avec beaucoup de soin. Les trois jeux de démonstration s'interrompent chaque fois qu'on presse sur la détente. En effet, un balayage d'écran continuel, indispensable quand on veut se servir d'un crayon optique, entraînerait un ralentissement inconcevable du déroulement de la partie. Le logiciel arrête donc tout à chaque appui sur la détente, vérifie si la cible s'aligne sur la position du fusil et, après seulement, relance l'action. En théorie tout cela ne devrait nécessiter qu'un sous-programme très court : mais ce n'est apparemment pas le cas, comme on peut s'en convaincre en observant les programmes.

Sur un ordinateur comme le BBC Micro, la puce vidéo est capable de traiter les messages en provenance d'un crayon optique, ce qui pourrait simplifier beaucoup les choses; pourtant il n'existe encore aucune version du fusil lumineux prévue pour cet appareil. Le Commodore 64 est lui aussi équipé d'un tel système. Mais le ZX Spectrum, sur lequel le dispositif a été testé, en est dépourvu, et c'est peut-être pourquoi il faut si longtemps pour calculer la position d'écran chaque fois qu'on presse sur la détente.

Tir dans la nuit

La cellule photoélectrique du « fusil lumineux » détecte le mouvement des points du balayage qui vont inscrire l'image sur l'écran. Le système informatique suit continuellement le balayage-écran de façon à connaître en permanence sa position; quand le fusil signale qu'il a détecté la position du système convertit cette position en coordonnées X et Y et les compare à celles de la cible.

Ouverture de la chasse

High Noon (Pleine lune) est le meilleur des trois programmes utilisant le fusil. L'animation est bonne et le chasseur peut tirer de façon sérieuse. Groose Shoot (Tir aux faisans) et Shooting Gallery (Tir forain) n'offrent qu'une seule possibilité : tirer avant que la cible ne disparaisse de l'écran.

Ian McKinnell

Définition de caractères

Le jeu de caractères graphiques de Commodore est évolué, mais il est souvent nécessaire de créer certains caractères spéciaux, ou même de redéfinir le jeu de caractères entier.

La création de caractères sur le Commodore 64 est assez compliquée; le BASIC Commodore n'offre aucune commande spéciale; l'opération doit donc être menée à l'aide de PEEK et de POKE pour accéder au contenu de la mémoire et pour le modifier.

Le jeu de caractères réside dans un bloc de ROM commençant à l'adresse 53248. Chaque caractère apparaît à l'écran sous la forme d'une configuration de points dans une matrice de huit points par huit; pour définir cette configuration de 64 points, on doit utiliser 64 bits (8 octets). Les 8 octets occupant les adresses 53248 à 53255 décrivent le caractère « a », le premier du jeu; il a un code écran 0, ce qui signifie que si vous écrivez (POKE) la valeur 0 dans l'un des octets de la mémoire RAM vidéo, ce caractère apparaîtra à l'écran. Les 8 prochains octets, de 53256 à 53263 décrivent « A » (code écran 1), et ainsi de suite.

Le jeu de caractères ROM partage son espace d'adressage en mémoire avec des dispositifs d'entrée/sortie comme des unités cassette ou des lecteurs de disquettes. Normalement, l'UC 6510A traite cet espace mémoire comme une zone d'entrée/sortie, mais il peut être programmé pour le considérer comme l'emplacement du jeu de caractères. Cela peut sembler étrange, mais l'UC n'a normalement pas à trouver les définitions de caractères dans la ROM et à les envoyer sur écran. Cette tâche est confiée à une autre puce placée sous le contrôle de l'UC. Le contenu de l'adresse 1 détermine le statut des opérations d'E/S, et le bit 2 de cette adresse a une fonction de bascule qui définit comment l'UC doit traiter la ROM du jeu de caractères. Si ce bit est mis à zéro, l'UC trouve les dispositifs d'E/S qui occupent l'espace. Les autres bits de l'adresse 1 ont des fonctions spéciales similaires de commande du système; nous devons donc être prudents pour ne pas les modifier en changeant la valeur du bit 2. La meilleure façon d'y arriver est d'utiliser les opérateurs ET et OU.

Supposons que l'adresse 1 contienne les éléments suivants :

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	1	1	1	1

Nous désirons mettre le bit 2 à zéro. Nous pourrions le faire en calculant la valeur décimale de 01101011, et l'écrire dans l'adresse 1; mais cela ne fonctionne que si nous savons que le contenu antérieur de l'adresse 1 était 01101111. Il est préférable de modifier le bit 2 en utilisant AND et PEEK. La commande suivante lit l'adresse 1, établissant ainsi son contenu initial, relie cette valeur à 251 (11111011 binaire) avec l'opérateur AND et écrit le résultat dans l'adresse 1 :

```
POKE 1, PEEK(1) AND 251
```

L'effet de cette commande peut être illustré ici :

Bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	1	1	1	= contenu initial
AND	1	1	1	1	1	0	1	1	= 251 binaire
	0	1	1	0	1	0	1	1	= Résultat de l'opération AND sur chaque paire de bits

Quelle que soit la valeur initiale du bit 2, l'opérateur AND couplant cette valeur à zéro produira toujours un résultat de zéro; l'opérateur AND couplant les autres bits de l'adresse avec 1 reproduit simplement leurs valeurs originales. Le nombre binaire 11111011 (251, en décimal) est appelé un masque, et il sert ici de masque AND.

Pour mettre le bit 2 à 1 sans affecter les autres bits, nous utiliserons la commande suivante :

```
POKE 1, PEEK(1) OR 4
```

Bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	0	1	1	= contenu initial
OR	0	0	0	0	0	1	0	0	= 4 binaire
	0	1	1	0	1	1	1	1	= Résultat de l'opération OR sur chaque paire de bits

Cela garantit que BASIC n'écrasera pas notre jeu de caractères. Lorsque la copie est terminée, l'UC peut de nouveau adresser les dispositifs d'E/S, et le mécanisme d'interruption repart.

La dernière opération est d'obliger la puce de traitement d'écran à utiliser notre jeu de caractères et non plus le jeu du système stocké en ROM. Les bits 0 à 3 de l'adresse 53272 désignent l'adresse de départ du jeu de caractères, et le tableau suivant illustre comment le Commodore 64 interprète les valeurs de ces bits pour désigner des adresses particulières :

valeur décimale des bits 0 à 3	bits 3,2,1,0	adresse désignée
0	0000	0
2	0010	2048
4	0100	4096
6	0110	6144
8	1000	8192
10	1010	10240
12	1100	12288
14	1110	14336

La valeur du bit 0 dans ce registre n'a pas d'importance, tandis que les bits 4 à 7 commandent d'autres fonctions, et ne doivent pas être modifiés. Nous utiliserons 11110000 (240 décimal) comme un masque AND à cet effet, et 00001110 (14 décimal) comme un masque OR pour faire en sorte que ce registre désigne l'adresse 14336 — l'adresse de départ de notre jeu de caractères :

```
POKE 53272,(PEEK(53272) AND 240) OR 14
```

Utilisons maintenant une boucle FOR...NEXT pour effectuer la copie.

Pendant qu'un programme copie le jeu de caractères ROM dans la RAM, l'UC ne peut gérer les interruptions des dispositifs d'E/S. Le clavier, par exemple, interrompt l'UC à chaque soixantième de seconde, afin qu'elle puisse repérer toute pression sur une touche. Ces interruptions sont déclenchées par le rythmeur du système. Si l'UC était interrompue par un dispositif d'E/S pendant que le jeu de caractères occupe l'espace E/S en ROM (comme c'est le cas pendant la copie), le système subirait alors un arrêt anormal, et la machine ne pourrait être réamorcée qu'en mettant la machine hors tension puis de nouveau sous tension. Heureusement, nous pouvons interdire le mécanisme d'interruption en mettant à 0 le bit 0 de l'adresse 56334; les autres bits de cette adresse ne doivent pas être modifiés, la commande POKE logique suivante doit donc être utilisée :

```
POKE 56334, PEEK(56334) AND 254
```

Dès que les interruptions sont interdites, et que l'UC a reçu l'instruction de chercher le jeu de caractères dans la ROM, la copie peut commencer.

Supposons que nous désirions copier les 64 caractères allant de « @ » à « ? » — codes écran 0 à 63 — nous devons alors copier les 512 adresses (8 × 64 = 512) commençant à 53248 dans un bloc de RAM adéquat. Diverses zones peuvent être utilisées; ici nous avons choisi un bloc qui commence à l'adresse 14336. Cela devrait normalement être la zone réservée au programme BASIC mais nous pouvons le protéger en abaissant le pointeur du haut de la mémoire à 56 :

```
POKE 56,32
```

Chaque caractère du jeu est conçu sur une matrice huit points par huit. Chaque ligne de la matrice est interprétée comme un nombre binaire (les points qui sont illuminés sont représentés par des uns, les points qui n'apparaissent pas sur l'écran sont représentés par des zéros) et nécessite donc un octet de stockage, et le caractère entier de huit lignes requiert huit adresses consécutives en mémoire. L'adresse de départ d'un octet décrivant un caractère peut être calculée à partir de l'adresse de départ du bloc, et du code écran du caractère particulier, ainsi :

Début du caractère = 14336 + 8 × (code écran)

Dès que les adresses des octets qui définissent un caractère sont connues, nous pouvons écrire de nouvelles valeurs dans ces octets, changeant ainsi les configurations de points apparaissant sur l'écran lorsque le caractère est imprimé.

```
130 :
140 REM *** COPIE DU JEU DE CARACTERE ROM ***
150 POKE56,32
    : REM ABAISSER LE HAUT DE LA MEMOIRE
160 POKE56334,PEEK(56334)AND254
    :REM MISE A L'ARRET DU RYTHMEUR
    D'INTERRUPTION
165 POKE1,PEEK(1)AND251
    :REM PASSER A LA ROM CARACTERES
170 FOR I=0 TO 511
    :REM COPIE
180 POKE14336+I,PEEK(53248+I)
    :REM 64
190 NEXT I
    :REM CARACTERES
200 POKE1,PEEK(1)OR4
    :REM REPASSER AUX E/S
210 POKE56334,PEEK(56334)OR1
    :REM REMETTRE EN FONCTION LE RYTHMEUR
    D'INTERRUPTION
220 POKE 53272,(PEEK(53272)AND240)
OR14:REM POSER LE POINTEUR DE CARACTERE
230 REM *** COPIE TERMINEE ***
240 :
250 :
300 REM *** ARTHUR L'ATHLETE ***
310 FOR I=14552TO14552+31
    REM:LECTURE
320 READ A:POKEI,A:NEXT I
    :REM DONNEES CAR
330 PRINTCHR$(147)
    :REM EFFACER L'ECRAN
340 POKE55338, 14:POKE55378,14
    :REM COULEUR, BLEU CLAIR
350 POKE 1066,27:POKE1106,28
    :REM BRAS LEVES
360 FOR I=1TO500:NEXT I
    REM: BOUCLE DE TEMPORISATION
370 POKE1066,29:POKE1106,30
    REM: BRAS BAISSES
380 FOR I=1TO500:NEXT I
    REM: BOUCLE DE TEMPORISATION
390 GOTO 350
480:
490:
500 REM *** DONNEES BRAS LEVES ***
510 DATA129,153,189,153,66,60,60,60
520 DATA60,60,36,36,66,66,195,0
530 REM *** DONNEES BRAS BAISSES ***
540 DATA0,24,60,24,0,126,189,189
550 DATA189,189,36,36,36,36,102,0
```

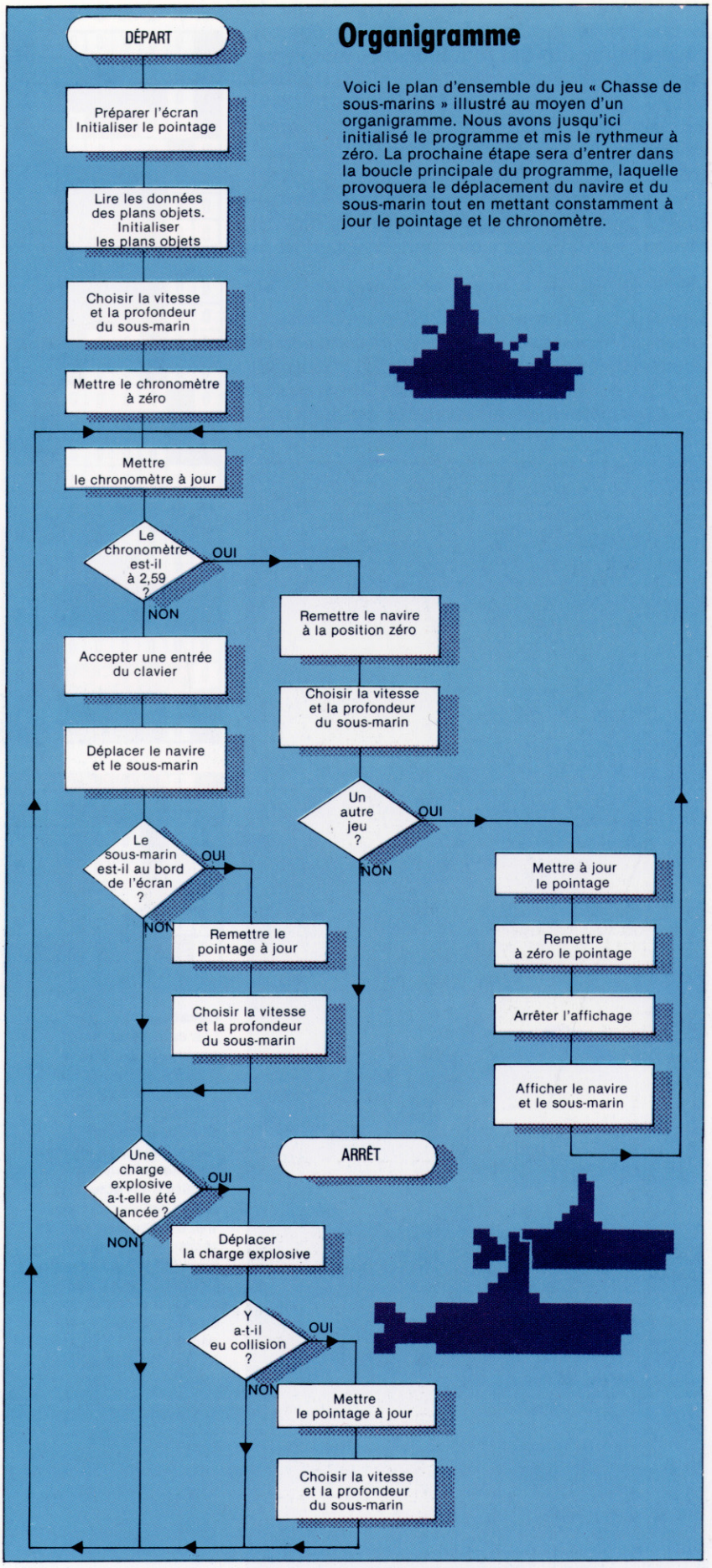
128	64	32	16	8	4	2	U	DÉCIMALE
1	0	0	0	0	0	0	1	129
1	0	0	1	1	0	0	1	153
1	0	1	1	1	1	0	1	189
1	0	0	1	1	0	0	1	153
0	1	0	0	0	0	1	0	66
0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60
0	0	1	1	1	1	0	0	60
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	1	0	0	0	0	1	0	66
0	1	0	0	0	0	1	0	66
1	1	0	0	0	0	1	1	195
0	0	0	0	0	0	0	0	0

128	64	32	16	8	4	2	U	DÉCIMALE
0	0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0	24
0	0	1	1	1	1	0	0	60
0	0	0	1	1	0	0	0	24
0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0	126
1	0	1	1	1	1	0	1	189
1	0	1	1	1	1	0	1	189
1	0	1	1	1	1	0	1	189
1	0	1	1	1	1	0	1	189
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	0	1	0	0	1	0	0	36
0	1	1	0	0	1	1	0	102
0	0	0	0	0	0	0	0	0

Construction
Les caractères sont construits sur une matrice de points huit par huit, définie dans huit octets consécutifs. Chaque ligne du caractère est interprétée comme un nombre binaire à un octet (les points représentent des 1 et les espaces des zéros). Pour être utilisés dans les programmes du Commodore 64, ces nombres binaires doivent être convertis en valeurs décimales.
(Cl. Liz Dixon.)

Organigramme

Voici le plan d'ensemble du jeu « Chasse de sous-marins » illustré au moyen d'un organigramme. Nous avons jusqu'ici initialisé le programme et mis le rythmeur à zéro. La prochaine étape sera d'entrer dans la boucle principale du programme, laquelle provoquera le déplacement du navire et du sous-marin tout en mettant constamment à jour le pointage et le chronomètre.



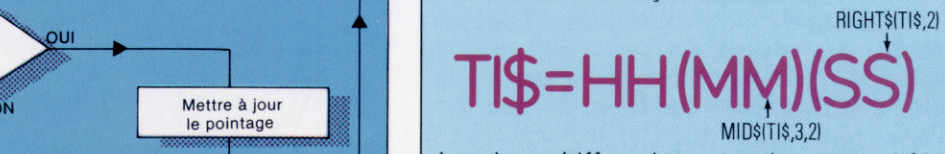
Programme « Chasse aux sous-marins »

Le Commodore 64 possède une horloge interne qui peut être utilisée pour chronométrer les programmes BASIC. L'horloge a six chiffres, un peu comme une montre numérique, représentant les heures (00-23), les minutes (00-59), et les secondes (00-59). L'horloge peut être lue à partir du BASIC au moyen de la variable de chaîne TI$. La valeur de TI$ donne le temps qui s'est écoulé depuis la mise sous tension de l'ordinateur; mais il est possible de la régler à une autre valeur à tout moment. Le court programme suivant illustre le fonctionnement de l'horloge.

```
10 REM *** HORLOGE ***
20 PRINT CHR$(147)      :REM EFFACER L'ÉCRAN
30 TI$ = « 000000 »     :REM METTRE L'HORLOGE A ZÉRO
40 PRINT CHR$ (145); TI$ :REM IMPRIMER LA VALEUR
                          ACTUELLE DU CHRONOMÈTRE
50                      :REM CHR$ (145) = CURSEUR EN
                          HAUT
60 GOTO 40
```

Le programme est exécuté dans une boucle continue, affichant le chronomètre jusqu'à ce que vous appuyiez sur la touche Run/Stop.

Le programme de jeu que nous écrivons doit afficher une horloge et mettre fin au jeu lorsque les trois minutes sont écoulées. L'horloge du jeu ne requiert donc que les portions secondes et minutes de TI$. En utilisant les fonctions de chaîne, nous pouvons fractionner TI$ de la façon suivante :



Les deux chiffres des secondes peuvent être extraits par RIGHT$(TI$,2), et les chiffres des minutes peuvent être isolés par MID$(TI$,3,2).

La principale boucle du programme de notre jeu commence à la ligne 200 et se termine à la ligne 390. Chargez le sous-programme déjà écrit dans la dernière section et ajoutez ces lignes :

```
140 TI$=«000000»
150 :
160 :
200 REM **** BOUCLE PRINCIPALE ****
205 :
210 REM ** CHRONOMÈTRE **
220 PRINT CHR$ (19);:TAB(14) CHR$(5) «TEMPS»;
    MID$(TI$,3,2);«:»;RIGHT$(TI,2)
225 IF VAL(TI$)>259 THEN 400:REM FIN DU JEU :
390 GOTO 200:REM RECOMMENCER LA BOUCLE PRINCIPALE
400 END
```

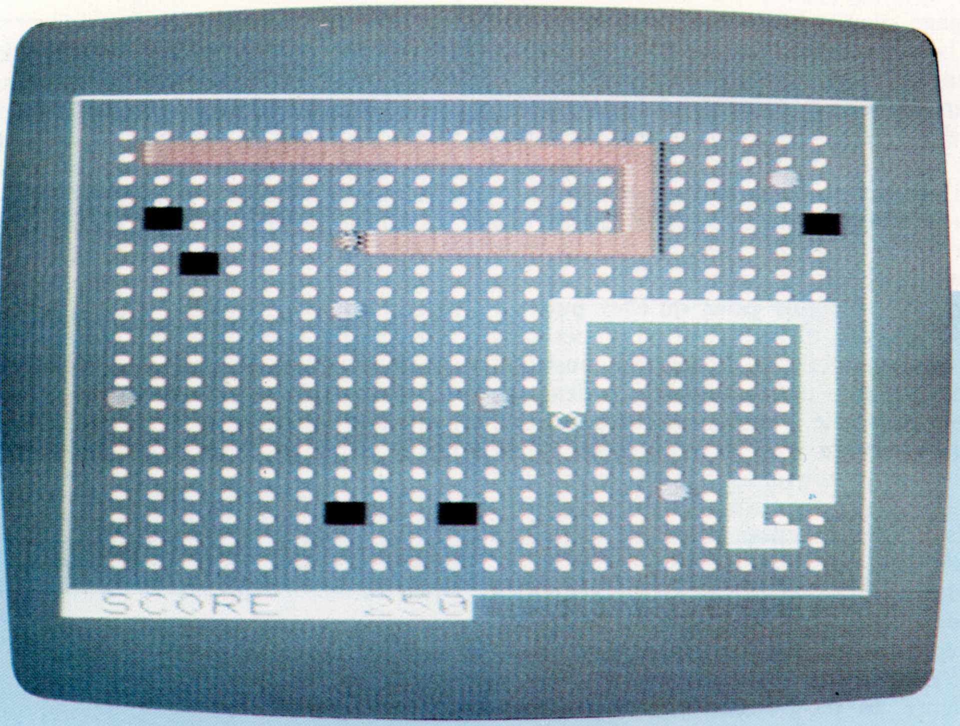
La ligne 140 remet à zéro l'horloge au début du programme. La ligne 220 imprime la valeur actuelle de l'horloge en minutes et en secondes, séparées par un deux points. TAB(14) insère 14 espaces avant l'impression de l'horloge, la plaçant ainsi au milieu de l'écran. CHR$(5) colore les caractères en blanc. La ligne 225 convertit TI$ en une quantité numérique afin de pouvoir tester sa valeur. Si le temps du jeu a dépassé 2 minutes et 59 secondes, le jeu prend fin.

Le piège

Attention à l'électricité. C'est elle qui forme le piège. A vous de l'éviter. Alastair Gourlay a conçu ce jeu en basic pour le micro-ordinateur Vic-20.

Dans ce jeu, vous vous déplacez dans un carré en laissant une trace derrière vous. L'ordinateur contrôle un robot qui laisse également une trace derrière lui. Si vous touchez les limites du carré ou si vos coupez une trace vous serez électrocuté.

Les points se comptent de la manière suivante :
si le robot est stoppé (le cercle vert) : 200 points;
si vous passez sur un point mauve : 50 points;
si vous passez sur un point blanc : 10 points.
La vitesse du jeu augmente à chaque partie.




```
10 RESTORE
13 GOSUB3000:GOSUB2000
15 X=RND(-TI):DEFFNR(X)=(PEEK(7663+X)=0)
20 D(0)=1:D(1)=22:D(2)=-1:D(3)=-22
30 L=150+(KJ$="J")*50
40 PP=46:PC=437:DP=1:DC=2:SC=7680:CL=38400:POKE36878,15
45 GOSUB1000
50 IFKJ$="J"THEN62
55 GETA$:POKE198,0
60 ND=22*((A$="P")-(A$="."))+(A$="L")-(A$=":"):IFND<>0THENDP=ND
61 GOTO70
62 SYS7600:X1=0:FORX=0TO3:X1=X1-FNR(X):NEXT:IFX1>1THEN70
65 ND=22*(FNR(0)-FNR(1))+FNR(2)-FNR(3):IFND<>0THENDP=ND
70 NP=PP+DP:X1=PEEK(SC+NP):IFX1<>46THEN700
75 POKE36874,128:S=S+10
80 POKESC+PP,160:POKECL+PP,2:POKESC+NP,42:POKECL+NP,2:PP=NP
85 PRINT" SCORE "S"";:POKE36874,0:POKE36876,0
90 NC=PC+D(DC):IFPEEK(SC+NC)<>46ORRND(1)<.15THEN200
100 POKESC+PC,160:POKECL+PC,5:POKESC+NC,15:POKECL+NC,5:PC=NC
110 FORX=0TOL:NEXT:GOTO50
200 FORX=1TO19:DC=INT(RND(1)*4):IFPEEK(SC+PC+D(DC))=46THENNC=PC+D(DC):GOTO100
210 NEXT
220 S=S+200:L=L/2
230 FORX=1TO4:FORY=128TO160:POKE36875,Y:NEXT:NEXT:POKE36875,0:POKE198,0:GOTO40
700 IFX1=81THENPOKE36876,200:S=S+50:GOTO80
800 POKE36877,250:FORX=1TO2000:NEXT:POKE36877,0
810 PRINT"          TERMINE          ":FORX=1TO5000:NEXT:POKE198,0:S=0:GO
TO10
1000 PRINT"┌──────────────────────┐";:FORX=1TO20:PRINT"│....................│"
;:NEXT
1010 PRINT"└──────────────────────┘";
1020 FORX=1TO5:X1=INT(RND(1)*20)+1+(INT(RND(1)*20)+1)*22
1030 POKE7680+X1,160:POKE38400+X1,0:NEXT
1040 FORX=1TO5:X1=INT(RND(1)*20)+1+(INT(RND(1)*20)+1)*22
1050 POKE7680+X1,81:POKE38400+X1,4:NEXT:POKESC+46,46:POKESC+437,46
1060 POKECL+46,1:POKECL+437,1:RETURN
2000 POKE36879,25
2010 PRINT"        PIEGE"
2020 PRINT"   DEPLACEZ VOUS EN   UTILISANT LE MANCHE A BALAIS OU-"
2030 PRINT"            PL*:I."
2040 PRINT"SCORE-":PRINT".-10 PTS":PRINT"-50 PTS":PRINT"ROBOT STOPPE-200 P
TS"
2960 PRINT" TAPEZ UNE TOUCHE"
2970 GETA$:IFA$=""GOTO2970
2980 POKE36879,110:IFKJ$<>""THENRETURN
2990 PRINT"CLAVIER OU MANCHE A   BALAIS?":PRINT"   (C/M)"
2991 GETCM$:IFCM$<>"C"ANDCM$<>"M"THEN2991
2992 RETURN
3000 POKE51,176:POKE52,29:POKE55,176:POKE56,29
3010 FORX=7600TO7661:READA:POKEX,A:NEXT
3020 RETURN
3030 DATA169,0,141,19,145,169,127,141,34,145,173,17,145,141,238,29,169,4,45,238,
29
3040 DATA141,239,29,169,8,45,238,29,141,240,29,169,16,45,238,29,141,241,29,169,3
2
3050 DATA45,238,29,141,243,29,173,32,145,41,128,141,242,29,169,255,141,34,145,96
```

Routine générale

Pour faire des programmes translatables, en langage machine, il faut utiliser des symboles et des labels au lieu d'adresses et de valeurs absolues. Nous étudions encore quelques pseudo-instructions d'assembleur.

Le langage d'assemblage étant avant tout un simple langage de programmation composé de commandes « primitives » que peut gérer l'UC, vous serez amené à écrire et récrire constamment des fragments de programmes pour effectuer les mêmes tâches qui sont impliquées dans le jeu d'instructions d'un langage évolué.

Deux problèmes majeurs apparaissent. Le premier est la difficulté d'écrire des routines importantes, et souvent longues, d'une manière suffisamment générale pour qu'elles puissent être insérées dans différents programmes sans ajustement ou réécriture. Le second consiste à écrire des routines utiles qui ne dépendent pas d'un seul ensemble d'emplacements mémoire, mais qui puissent être « translatées » en mémoire par un nouvel assemblage avec différentes adresses ORG, et qui accomplissent exactement la même fonction dans les différents emplacements.

Les deux problèmes concernent la généralité et la portabilité (questions familières aux programmeurs de BASIC) et sont résolus de la même façon : en utilisant des variables pour passer des valeurs du programme général au sous-programme ; en employant des variables locales dans les sous-programmes, qui soient indépendantes du contexte général ; et en évitant l'utilisation de quantités absolues (constantes numériques ou de chaîne) et des numéros de ligne.

En programmation d'assemblage, nous nous sommes familiarisés avec l'idée d'emplacements mémoire, homologues des variables BASIC — les programmes opèrent sur le contenu des emplacements, quel que soit ce contenu, de la même manière qu'un programme BASIC agit sur le contenu de ses variables. Malheureusement, nous nous référions aux emplacements mémoire par leurs adresses absolues, habitude pratique au début, mais à laquelle il faut renoncer pour gagner en généralité. La solution consiste à utiliser des symboles au lieu d'adresses et de valeurs absolues, et à se servir de toute la série de formes symboliques offerte par les pseudo-op d'assembleur comme des équivalents des variables et des numéros de lignes de programme. Nous avons vu des exemples de ces deux cas. Considérons par exemple le programme suivant :

6502			Z80		
DATA1	EQU	$12	DATA1	EQU	$12
DATA2	EQU	$79	DATA2	EQU	$79
	LDA	DATA1		LD	A,(DATA1)
LOOP	ADC	DATA2		ADC	A,(DATA2)
	BNE	LOOP		JR	NZ,LOOP
	RTS			RET	

Nous avons ici deux espèces de symboles, deux valeurs et un label, utilisées toutes deux comme opérandes d'instructions en langage d'assemblage. De ce fait, le fragment de programme est à la fois général et translatable. Les seules quantités absolues sont les valeurs de DATA1 et DATA2.

D'autres pseudo-op n'ont pas encore été examinés. En particulier, DB, DW et DS (quoique, comme ORG et EQU, ils puissent différer d'un assembleur à l'autre). Ces trois pseudo-instructions, qui signifient « définir octet » (Define Byte), « définir mot » (Define Word) et « définir stockage » (Define Storage), nous permettent d'initialiser et d'allouer des emplacements mémoire, comme ainsi :

			ORG	$D3A0
D3A0	5F	LABL1	DB	$5F
D3A1	CE98	LABL2	DW	$98CE
D3B3		LABL3	DS	$10
D3B3		DATA1	EQU	LABL3

Table des symboles :
LABL1 = D3A0:LABL2=D3A1:LABL3=D3A3
DATA1 = D3A3
ASSEMBLY COMPLETE – NO ERRORS

Dans ce listage d'assemblage (sortie d'un programme assembleur) nous voyons en bas, pour la première fois, une table des symboles comprenant les symboles définis dans le programme et les valeurs qu'ils représentent. Il y a plusieurs points importants à relever dans ce fragment. Tout d'abord, à la ligne LABL1, le pseudo-op DB est utilisé. Le listage nous montre que la pseudo-instruction ORG a donné l'adresse $D3A0 à LABL1, et la table des symboles confirme cela. L'effet de DB consiste à mettre la valeur $5F dans l'octet adressé par LABL1 — de sorte que l'emplacement mémoire $D3A0 est initialisé avec la valeur $5F, comme nous pouvons le voir dans la colonne langage machine du listage.

Ensuite, LABL2 représente l'adresse $D3A1. Toutefois DW a pour effet d'initialiser un « mot » (deux octets consécutifs) de stockage, de sorte que la valeur $98CE est stockée aux emplacements $D3A1 et $D3A2 sous la forme lo-hi, ce qu'on peut

voir clairement dans la colonne langage machine. Puisque DW convertit automatiquement ses opérandes en forme lo-hi, il est surtout utilisé pour initialiser les emplacements « pointeurs » avec des adresses. LABL2, ou l'emplacement $D3A1, doit être une telle adresse — indiquant l'emplacement $98CE.

La troisième chose à considérer est que l'instruction DS $10 a pour effet d'additionner $10 au compteur de programme. C'est plus évident dans la table des symboles que dans le listage — LABL3 représente l'emplacement $D3A3 (l'emplacement suivant l'instruction précédente), bien que le listage montre que sa valeur est $D3B3. C'est en fait l'adresse de l'instruction suivante après DS, de sorte que DS $10 a réservé un bloc de 16 octets (de $D3A3 à $D3B2 inclus) entre une instruction et la suivante. C'est un processus un peu analogue à l'insertion de lignes REM dans un programme BASIC afin de créer des espaces inutiliisés dans la zone texte, sur lesquels on puisse faire POKE et PEEK comme dans une zone de programme langage machine.

Enfin, la dernière instruction utilise EQU pour égaler un symbole à la valeur d'un autre, de sorte que DATA1 a la valeur $D3A3 (valeur de LABL3). C'est une autre source de confusion possible. LABL3 est la représentation symbolique de l'adresse $D3A3, de sorte que DATA1 EQU LABL3 signifie « le symbole DATA1 doit avoir mêmes sens et valeur que le symbole LABL3 ». Le fait que l'instruction DB a égalé le contenu de $D3A3 à $5F n'a pas de signification pour les symboles LABL3 et DATA1. Se convaincre de cette distinction entre une adresse et son contenu, c'est l'une des plus grandes difficultés des premiers stades de la programmation en langage d'assemblage.

Au premier abord, la pseudo-instruction DB semble faire double emploi avec EQU, mais ce n'est pas le cas. LABL1 signifie « l'emplacement $D3A0 », et DB $5F a initialisé cet octet avec la valeur $5F mais, bien que la valeur de LABL1 soit maintenant fixée, le contenu de l'emplacement qu'elle symbolise peut être modifié à tout instant (en y stockant le contenu de l'accumulateur ultérieurement dans le programme, par exemple). De même, DATA1 est à présent un symbole dont la valeur est fixée par l'instruction EQU; sa valeur ne peut être changée par l'exécution du programme. De plus, LABL3 indique le début d'une zone de données de 16 octets, dont le contenu peut être modifié dans le programme, mais LABL3 lui-même est inchangeable.

Cela introduit, mais n'épuise pas, les possibilités des nouveaux pseudo-op. Considérons cette nouvelle version du fragment précédent :

			ORG	$D3A0
D3A0	4D4553	LABL1	DB	"MESSAGE1"
D3A9	CE98	LABL2	DW	$98CE
D3BB		LABL3	DS	$10
D3BB		DATA1	EQU	LABL3

Table des symboles :
LABL1 = D3A0:LABL2=D3A9:LABL3=D3AB
DATA1 = D3AB
ASSEMBLY COMPLETE – NO ERRORS

L'instruction DB a une chaîne, "MESSAGE1", comme opérande, et l'assembleur a initialisé les emplacements de $D3A0 à $D3A8 avec les valeurs ASCII des caractères à l'intérieur des guillemets simples. Cela découle de l'examen de la colonne d'adresses dans le listage, et c'est partiellement confirmé par la colonne langage machine — le contenu des trois octets de $D3A0 à $D3A2 est égal à $4D, $45 et $53 qui sont les codes ASCII hex pour « M », « E » et « S ».

C'est là une facilité considérable, non seulement parce qu'elle décharge le programmeur de la tâche de traduire les messages et données en listes de codes ASCII, mais aussi parce qu'elle rend le listage beaucoup plus aisé à lire et implique la possibilité d'obtenir une sortie à l'écran de vos programmes en langage d'assemblage. Cette dernière possibilité est particulièrement intéressante car jusqu'à présent nous nous sommes contentés de stocker des résultats en mémoire et de les inspecter en utilisant le programme moniteur. Naturellement, nous explorerons le maniement d'écran dans ce cours, mais il nous faut encore étudier des aspects du langage d'assemblage avant d'y arriver. Si toutefois vous vous posez des questions au sujet du stockage des résultats en mémoire et si vous comprenez déjà que les affichages de mémoire sur l'écran ne sont, en effet, que des zones de mémoire, alors il est possible que vous voyiez un mode d'adressage de l'écran à partir d'un programme.

Exercices

1. Le premier fragment de programme dans le texte principal utilise le pseudo-op DS pour réserver $10 octets de mémoire à partir de l'adresse représentée par le label LABL1. Écrivez un programme en langage d'assemblage qui stocke les nombres de $0F à $00 par ordre décroissant dans ce bloc, un nombre par octet. Cela peut être fait au moyen d'une boucle, et les techniques d'adressage indexé, pour lesquelles vous devrez utiliser les instructions DEX (décrémente le registre X) ou DEC(IX+0) (décrémente IX). La boucle devra se poursuivre tant que le registre d'index n'a pas eu pour effet de mettre le drapeau de zéro; il faut donc utiliser les instructions de branchement BNE ou JR NZ.

2. En utilisant les techniques de l'exercice précédent, écrivez un programme pour copier le message stocké en LABL1 par le pseudo-op DB (voir second fragment de programme dans le texte principal) sur un bloc de mémoire commençant à l'adresse stockée en LABL2 par le pseudo-op DW. L'adresse $98CE peut ne pas convenir à votre ordinateur; changez alors l'initialisation, mais le programme devrait marcher pour toute adresse et pour toute longueur de message. Pour mettre cela en œuvre, votre programme devra soit utiliser le nombre de caractères dans le message comme compteur de boucle, soit pouvoir reconnaître la fin du message — vous pouvez mettre un astérisque, par exemple, comme dernier caractère de message.

L'aspect le plus important de cette nouvelle facilité de DB est qu'elle confère à LABL1 le statut d'une variable chaîne BASIC : lorsque nous écrivons en BASIC :

```
200 LET A$="MESSAGE1"
```

nous créons en fait un pointeur pour le début de la table d'octets contenant les codes ASCII pour « M », « E », « S », etc. Chaque fois que l'interpréteur BASIC rencontre une référence à A$, il cherche dans sa propre table des symboles l'adresse qu'elle indique — c'est-à-dire l'emplacement de départ du contenu de A$. De même, dans notre programme en langage d'assemblage, nous pouvons traiter LABL1 comme l'équivalent de A$, étant donné que nous avons déjà écrit un fragment de programme qui nous permet de manipuler une table utilisant l'adressage indexé.

Les pseudo-op nous permettent alors de supprimer de nos programmes les adresses et valeurs absolues, et de les remplacer par des symboles. Cela a pour effet de réduire les problèmes de portabilité et de relocalisation. Ce qu'il nous faut maintenant, c'est de pouvoir accéder à ces modules portables à partir du programme principal. Autrement dit, nous avons besoin de l'équivalent en langage machine de la commande BASIC GOSUB.

Cette instruction existe, bien sûr : c'est respectivement JSR et CALL sur 6502 et Z80. Toutes deux requièrent une adresse absolue (qui peut être un label) comme opérande, et toutes deux ont pour effet de remplacer le contenu du compteur de programme par l'adresse qui forme leur opérande. L'instruction suivante à effectuer sera donc la première instruction du sous-programme ainsi adressé. L'exécution continue à partir de cette instruction, jusqu'à l'instruction de retour — respectivement RTS et RET. Cette commande a pour effet de remplacer le contenu en cours du compteur de programme par le contenu précédant immédiatement l'instruction JSR ou CALL. C'est exactement le mécanisme utilisé pour l'interpréteur BASIC lorsqu'il exécute et retourne après GOSUB. Il se comprend aisément comme cela, mais la question se pose de savoir comment l'ancien contenu du compteur de programme est rétabli lorsque l'instruction de retour est exécutée. Tout simplement, les instructions JSR et CALL déplacent d'abord le contenu du compteur de programme sur la pile, avant de le remplacer par l'adresse du sous-programme; et les instructions RTS et RET recherchent cette adresse sur la pile pour la remettre dans le compteur de programme. Quant à savoir ce qu'est la pile, tout cela vous sera expliqué dans quelque temps.

Jeu d'instructions

BEQ — **BRANCHEMENT EN ZÉRO** — Relatif — F0 — (2 octets) — **6502**

Le contenu du compteur de programme est décalé de la valeur de l'octet suivant l'opc.

EFFET SUR LE PSR

	S	V		B	D	I	Z	C
MSB								LSB

PAS D'EFFET

Exemple :

Adresse	Langage machine	Langage d'assemblage
8F00	F0 16	BEQ $16

		AVANT		APRÈS	
Compteur programme	lo	02		18	
	hi	8F		8F	
				F0	$8F00
				16	
					Mémoire programme

JR Z — **– SAUT RELATIF EN ZÉRO** — Relatif — 28 — (2 octets) — **Z80**

Le contenu du compteur de programme est décalé de la valeur de l'octet suivant l'opc.

EFFET SUR LE PSR

	S	Z		H		V	N	C
MSB								LSB

PAS D'EFFET

Exemple :

Adresse	Langage machine	Langage d'assemblage
8F00	28 16	JR Z,$16

		AVANT		APRÈS	
Compteur programme	lo	02		18	
	hi	8F		8F	
				28	$8F00
				16	
					Mémoire programme

INX — **– INCRÉMENTATION DU REGISTRE X** — Implicite — E8 — (1 octet) — **6502**

Le contenu du registre X est incrémenté de un.

EFFET SUR LE PSR

	S	V		B	D	I	Z	C
MSB	X						X	LSB

Exemple :

Adresse	Langage machine	Langage d'assemblage
F391	E8	INX

	AVANT	APRÈS	
PSR	????????	0???????	
X	FF	00	
		E8	$F391
			Mémoire programme

INC IX — **– INCRÉMENTATION DE IX** — Implicite — DD23 — (2 octets) — **Z80**

Le contenu de IX est incrémenté de un.

EFFET SUR LE PSR

	S	Z		H		V	N	C
MSB								LSB

PAS D'EFFET

Exemple :

Adresse	Langage machine	Langage d'assemblage
F391	DD23	INC IX

		AVANT		APRÈS	
Compteur programme	lo	FF		00	
	hi	E7		E8	
				DD	$F391
				23	
					Mémoire programme

Liz Dixon

Esprit de système

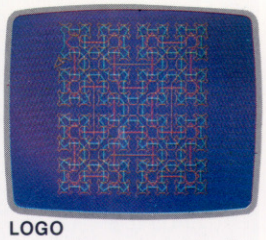
Depuis sa création, le système d'exploitation CP/M (Control Program for Microprocessors) est devenu une véritable norme en informatique. Son succès a bouleversé la vie de son concepteur, Gary Kildall.

Gary Kildall était membre de l'équipe de recherche qui mit au point, pour Intel, le microprocesseur 8080. Sa première version du CP/M, créée en 1974, avait pour but de fournir un compilateur PL/M — le premier langage évolué produit par la compagnie. Il y ajouta l'année suivante un éditeur (ED), un assembleur (ASM), ainsi qu'un programme de recherche d'erreurs (DDT). Il offrit le tout à Intel qui déclina la proposition : ce refus fut sans doute la plus grande chance de sa vie. Associé à Dorothy McEwan, il se mit à publier des revues pour passionnés d'informatique, et à vendre aux amateurs des copies de son système d'exploitation. Le succès fut immédiat.

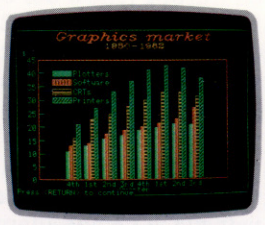
Volontairement ou non, Kildall avait en effet mis au point un programme qui apportait un début de solution au plus gros problème alors rencontré par la micro-informatique naissante : la compatibilité. Les trois micro-ordinateurs les plus répandus à la fin des années soixante-dix (le PET, l'Apple, le TRS Tandy) avaient des systèmes d'exploitation de disquettes tout à fait différents, et les firmes productrices de logiciels étaient contraintes de choisir un format ou un autre : il fallait donc récrire tout programme destiné à une machine particulière, puisqu'il ne tournait pas sur les deux autres. CP/M changea tout cela : son immense popularité le fit adopter par de nombreux constructeurs, dont les ordinateurs abritaient un 8080 d'Intel ou un Z80 de Zilog : le système permettait une manipulation aisée de l'écran, de l'imprimante, des disquettes, du clavier... Son succès en fit une norme de fait, pour laquelle on écrivait toujours plus de logiciels nouveaux, ce qui était une raison supplémentaire de recourir à lui.

CP/M était au départ l'abréviation de Control Program Monitor : ce titre d'allure modeste céda assez vite la place à celui de Control Program for Microprocessors. Dans un premier temps, seuls quelques utilisateurs choisis purent en bénéficier; en 1976, toutefois, Kildall se vit assailli de tant de demandes qu'il abandonna son poste de professeur d'informatique à l'université navale de Monterey pour fonder Digital Research à Pacific Grove, en Californie.

La nouvelle compagnie s'intéressa aux systèmes multi-utilisateurs à l'intention desquels elle lança MP/M, qui devait être pleinement compatible avec CP/M, mais qui ne connut jamais le même succès. Ce que devait savoir un programmeur (découpage des zones allouées aux utilisations, nature des diverses configurations, etc.) n'avait rien d'évident; de surcroît, le maniement des fichiers différait parfois de celui de CP/M. Enfin la baisse du prix de revient d'un microprocesseur rendait inutile le partage d'une même unité centrale entre plusieurs personnes. MP/M, bien que revu plusieurs fois, ne devint jamais populaire.



LOGO
Digital Research a abordé victorieusement le marché des langages de programmation. Son DR LOGO a, comme toutes les bonnes versions du LOGO, de remarquables possibilités graphiques.



Graphisme de gestion
GSX est un logiciel d'avant-garde destiné à rendre les applications graphiques transférables d'une machine à l'autre. C'est le cas de ce graphisme de gestion présenté ici. (Cl. Ian McKinnell.)

Digital Research Inc.

John Rowley, président de Digital Research Incorporated

Gary Kildall

Digital Research Inc., Massachusetts, U.S.A.

Digital Research

Digital Research obtint des fonds de plusieurs compagnies, spécialisées dans le « capital-risque », pour devenir en 1981 une véritable multinationale, particulièrement bien implantée en Europe (elle a des bureaux en Allemagne, en France et en Grande-Bretagne). A peu près à la même époque, elle fit tout son possible pour signer avec IBM un contrat et mettre au point le système d'exploitation du tout nouvel ordinateur individuel que « Big Blue » s'apprêtait à lancer. Microsoft finit cependant par l'emporter, mais Digital Research, ne s'avouant pas vaincue, a depuis adapté le CP/M afin qu'il soit compatible avec les microprocesseurs 8088 et 8086 d'Intel, et donc avec le PC et son système d'exploitation MS-DOS.

Elle est allée encore plus loin avec le Concurrent CP/M. Il fonctionne à l'inverse du MP/M, puisqu'il permet à plusieurs programmes d'être exécutés simultanément. Un particulier peut ainsi travailler à trois tâches différentes en même temps, et passer à volonté d'un tableur à un courrier électronique ou à un générateur d'états. Chaque écran correspondant — ou même une partie de celui-ci — peut être affiché au même moment grâce à des « fenêtres ». Les versions récentes du système devraient, elles aussi, accepter la plupart des programmes rédigés pour le DOS de l'IBM PC.

L'une des décisions les plus importantes de Digital Research (imitée en cela par bien d'autres compagnies) est de recourir exclusivement au langage C pour tous ses travaux de développement.

Il est sans doute justifié de dire qu'une véritable portabilité ne s'obtient que par le biais de langages très évolués. Mais Digital Research, qui cherche aussi à occuper le marché de l'informatique individuelle, a créé un département spécialisé et propose de nombreux langages à l'intention de divers micro-ordinateurs : Personal BASIC, Personal CP/M, ainsi qu'une version particulière de LOGO. CP/M-86 et Personal CP/M doivent être stockés en mémoire morte, et seront bientôt disponibles sur un microprocesseur Z80, grâce à un accord conclu avec Zilog. C'est là un bon moyen de prolonger la vie active de bon nombre de programmes CP/M « standard » déjà anciens : leur bas prix les mettra à la portée du simple particulier.

VIP et GSX sont deux autres systèmes riches de multiples potentialités, VIP est un « langage de commande » visuel bon marché, uniformément utilisé par le programmeur quel que soit le programme d'application en cours d'exécution. Il est possible de manipuler les mêmes données pour plusieurs programmes différents, et de les transférer de l'un à l'autre. De ce point de vue VIP est assez semblable au MacIntosh et à la Lisa d'Apple, mais se montre beaucoup moins gourmand en mémoire : il peut tourner sur tout ordinateur comportant plus de 50 K de RAM, et équipé d'une mémoire auxiliaire sur disquette de 150 K ou plus.

GSX est censé faire pour le graphisme ce que CP/M fait pour les disquettes. Il a recours à un ensemble de fonctions graphiques qui peuvent être mises en œuvre sur des matériels très différents : un programme GSX tournera sans modifications sur un écran couleur, noir et blanc, sur une imprimante ou une table traçante. Pourtant des difficultés subsistent : le système ne peut parvenir à la qualité des programmes rédigés pour un appareil spécifique, et souffre du manque de logiciels.

Bien qu'étant l'une des toutes premières firmes sur le marché de la micro-informatique, Digital Research n'entend pas se reposer sur ses lauriers. Outre des produits comme le LOGO, GSX ou VIP, elle entend imiter Microsoft et aborder le domaine prometteur des programmes d'application. Le succès jamais démenti du CP/M lui permet d'envisager l'avenir avec confiance.

aBc INFORMATIQUE

Pour classer et présenter dans votre bibliothèque les fascicules de votre collection, des reliures mobiles pratiques et élégantes sont en vente chez tous les marchands de journaux.
Dans chaque reliure, vous trouverez, dans l'enveloppe qui contient les deux lames métalliques de la reliure, un décalque portant les numéros 1 à 8, qui vous permettra de marquer vous-même le dos de chaque volume.
Pour relier les 12 fascicules qui composent un volume, vous devrez en retirer les couvertures sans endommager les agrafes métalliques. Le numérotage de la reliure puis la mise en place des fascicules doivent être effectués selon les instructions ci-dessous.

1. **Disposez à plat la reliure. Enlevez le papier de protection du décalque. Positionnez le décalque en faisant coïncider l'écran qui entoure le numéro choisi avec l'écran situé sur le dos de la reliure.**

2. **Avec la pointe d'un stylo à bille, frottez régulièrement le numéro à transférer, en exerçant une certaine pression et en débordant légérement.**

3. **Enlevez doucement le support : le numéro est reporté sur la reliure. Posez dessus le papier de protection du décalque et frottez largement avec un objet poli ou arrondi, de manière à assurer la parfaite adhérence des caractères transférés.**

4. **Retournez la reliure mobile. Introduisez, d'un côté seulement, dans les encoches pratiquées dans l'épaisseur de la couverture, l'une des extrémités des deux lames d'acier livrées avec la reliure.**

5. **Vérifiez le bon ordre des fascicules, puis passez les deux lames d'acier dans les agrafes supérieures et inférieures.**

6. **Introduisez les extrémités libres des lames d'acier dans les encoches correspondantes de la reliure.**

Ce système original, sans mécanisme visible, donne au volume l'apparence d'une reliure classique. Nous vous conseillons d'avoir toujours une reliure d'avance pour mieux protéger vos fascicules au fur et à mesure de leur parution.

Protégés par une élégante reliure, vos numéros d'aBc Informatique seront plus faciles à consulter

Pour classer, répertorier, protéger vos fascicules d'ABC Informatique, les Éditions Atlas vous proposent des reliures élégantes, sobres, qui s'insèrent parfaitement dans votre bibliothèque.
Chacune contient 12 fascicules, les maintient, les préserve. Un système simple, résistant, vous permet de les assembler facilement. Elles sont en vente en permanence chez votre marchand de journaux. Demandez-les !

Chaque reliure:

40 FF
295 FB
18 FS

Cod. N.M.P.P.: 6103

EDITIONS ATLAS